ENCYCLOPEDIA FOR THE TRS-80*

A library of useful information for your TRS-80*

Business
Education
Games
Graphics
Hardware
Home Applications
Interface
Tutorial
Utility

VOLUME **4**

ENCYCLOPEDIA for the TRS-80*

ENCYCLOPEDIA for the TRS-80*

VOLUME 4

wayne GREEN inc.
PETERBOROUGH NH 03458

FIRST EDITION
FIRST PRINTING DECEMBER 1981

Edited by Kate Comiskey and Katherine Putnam
Technical assistance by Jake Commander
and Kenniston W. Lord Jr., CDP
Illustrations by Howard Happ
Typeset by Karen Stewart

FOREWORD

The Biggest Difference

There are lots of arguments about which computer is the best. The answer to this question lies not in which hardware is best. That is really irrelevant, when you understand the field. The major value of any computer lies in the software and the information available for it. Hence this encyclopedia.

The TRS-80 is by no means the best computer on the market as far as its hardware is concerned, but with the support of *80 Microcomputing* magazine and this encyclopedia series, you have an almost unlimited source of information on how to use your computer—and of programs. With this information source the TRS-80 is by far the most valuable computer system ever built. No other computer, at any price, has anything approaching this amount of user information and programs available.

Most encyclopedias try to freeze everything at one time and are thus able to divide the material up alphabetically. This is a new kind of encyclopedia—a living one—with each new volume keeping you up to date on the very latest information on using your computer and the newest of programs.

Your computer can be a fantastic teaching device, a simulator, a way to play all sorts of fascinating games, a business aid, a scientific instrument, a control unit for machinery. . . . It is one of the most flexible gadgets ever invented. All of these applications are possible *if* you have the information and the programs. This encyclopedia will give you these.

To get the best use of your TRS-80, don't miss a single volume of the *Encyclopedia for the TRS-80.*

WAYNE GREEN
Publisher

CONTENTS

Please note: Before typing in any listing in this book, see Appendix A.

contents

BUSINESS

Mailing List for a Small Business
Business Forms—
The Statement

BUSINESS

Mailing List for a Small Business

by Charles P. Knight

There are hundreds of mailing list programs for the TRS-80, offering all sorts of advanced features, and usually, advanced price tags as well. These are fine if you have been in business a long time and have a customer list of moderate size, but they offer little advantage for the new business needing only an "occupant" mailing. Most new, small (1 or 2 employees) neighborhood businesses begin their advertising with circulars, but it is hard to determine whether the piece has been placed at the customer's door or down the nearest manhole cover.

The public library or city hall usually contains ample information about streets and valid house numbers to build a simple occupant mailing list for the particular area in which your business is located. Maillist is a program to manage this list while providing rapid ways to enter your data. The resulting labels will be addressed to OCCUPANT or any other string of up to 26 characters in length. Look at your latest Radio Shack flyer to see this type of list in use.

The program requires that you create a separate disk file every time there is a change in any data field on the label other than the actual house number. Since the street name, city, state, and zip code lines are stored only once per data file, and the occupant field is provided by the program, you can have an enormous number of individual addresses stored in a very small space on a disk. You will probably run out of file directory entries long before you run out of space on your diskette. These files will usually contain only one or two hundred house numbers and will only occupy a gran or two of disk space. NEWDOS80 users should specify DDGA = 3 at FORMAT time to overcome this difficulty.

Using the Program

The real splendor of this program is the ease with which data is entered. When you first run the program, you are greeted with a title logo and asked if you want instructions. After this, you reach the main menu where you may chose one of seven options. The first of these is the data entry option. Select this option and you will be asked to enter the city, state, and zip code for this file. You may input as many characters (including commas) as you wish, but the data must be applicable to all the addresses you will be entering for this file. If you wish to add a file already in memory, enter ADD-ON (use uppercase only), and the data in memory will be appended.

The previous entry will not be shown before the first data item is typed, but the information in the upper right corner of the display will be correct. Next, you will enter the name of the street. A new file has to be created when you change the street, city, or zip code.

Next, you are presented with an envelope drawn on the screen to simulate addressing a piece of mail. Notice that the complete address is already entered except for the house number. In the upper left of the screen, the last item entered is displayed. Before the first item is entered, XXXXX is displayed. The upper right corner keeps you notified of the number of entries you have made, and the number of entries you may still make before you must create a new file. You are also reminded that pressing @ will cause you to exit from this phase of the program. The message at the bottom of the screen reminds you that Microsoft BASIC will occasionally hang up to gather string space without warning. This is referred to as garbage collection and is rarely a problem in this program, but the presence of this message will keep the user from pressing the reset button and losing an hour's work for nothing. When you have entered all the house numbers on the street, enter @, and you will be prompted for a filespec for this file. I usually use the first eight characters of the street for this name and use the /EXT to denote the difference between street, boulevard, avenue, circle, etc. If, for some reason, you do not wish to write this data to disk just yet, enter @ for the filespec, and an error trap will return you to the menu without writing anything to disk. After the file is written, you will be returned to the menu, where you may enter data for another street, or elect to print labels, align the printer, end the program, write the file to disk again (for backup under another filespec, on a different drive or after sorting), or sort the street numbers.

The Sort Routine

House numbers are usually entered numerically and will not need sorting; however, I have provided a simple sort routine for those who like to enter data backward, but want it sorted forward. The routine, taken from the sort algorithm in David Lien's book *Learning Level II*, is a slow sort made slower by the constant displaying of the sorting process on the screen. The fact that the strings are sorted on their numeric values rather than their string values doesn't help either, but if they weren't, 1001 would come before 304. A typical 50-number sort takes only about a minute, so it isn't intolerable. The large SORT displayed on the screen during sorting helps tell at a glance what's going on just in case you're doing something else and want to check on the progress from across the room.

After sorting, you will probably want to write the sorted file to disk. Option 7 is provided for this purpose. On all writes to and reads from the disk, the items being read/written are dynamically displayed at the upper left-hand corner of the screen. By watching the data being read in from disk, you can really come to appreciate the speed of today's modern microcomputer.

Before printing labels, it is usually necessary to align them in the printer. This program provides a routine which outputs three strings of 24 Xs to allow vertical and horizontal centering of the label. When in this routine, each time you press the ENTER key, another label of Xs will be printed. Pressing the space bar returns you to the menu, where you may select option 3 to print out the labels for the entire file.

Always use option 6 to end the program. This option releases the string space cleared for data storage and leaves maximum memory available for the next program. This also wipes out all the stored data (but not the program), so be sure you have saved the file before exiting.

The Maillist Program

The program begins at line 80. First, 50 bytes of string storage space are cleared so that two thirds of available memory can be calculated in the next statement. If this were not done, the second time the program was run, fewer bytes would be cleared than the first time. Line 130 is the beginning of the program after initialization. Should you ever find yourself at BASIC ready and do not want to lose data in memory, type GOTO130 to get back to the menu without loss of data.

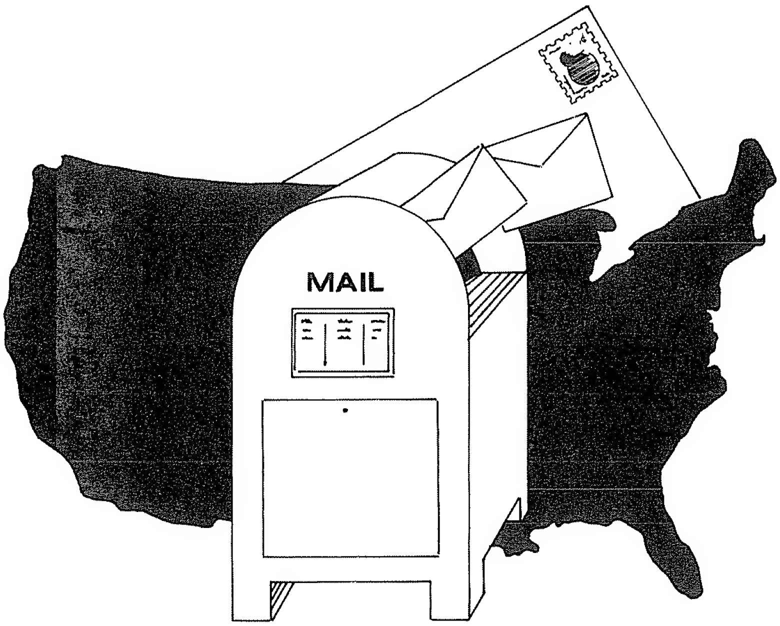
Lines 170 to 470 are the data entry section of the program, with line 340 being the entry point if only a file write is to be done. The statement at line 210 allows you to add on to a file already in memory, as noted above. Lines 350 to 470 contain the routine to write the data to disk. The first item written is the city, state, and zip code string (CZ$), and the next item is the street name string (ST$). The house numbers are stored individually. For those of you who use NEWDOS80 and compile your programs with Microsoft's BASIC compiler, there is no LINEINPUT# incompatibility here because CZ$ and ST$ are the only values that are LINEINPUT, and they are never long enough to span a sector. I've tried it, and it works every time with both the early and later versions of this compiler.

Lines 480 to 600 are the routine to load a file from disk. While loading a file, the information is dynamically displayed in the upper left corner of the screen. I have used this type of dynamic display throughout the program primarily because I like it, but it also assures the operator that something is going on and it can be a great time-saver when debugging programs.

Lines 610 to 720 contain the routine to print the labels. If you wish to stop printing, simply hold down the @ key, and you will return to the main menu. The label is displayed on the screen while it is being printed.

Lines 730 to 850 print a label consisting entirely of 24 Xs, making it easy to get the label centered in the printer correctly. Pressing the space bar returns you to the menu.

The remainder of the program consists of subroutines that are called by the main program. Beginning at line 890 is the data input routine. All data entered into the program come through this routine. The program passes the values S and E to the subroutine, informing it where to start and end the input on the screen. This limits the length of each item input to that desired by the calling routine. TT$ contains the data which is passed back to the calling routine, and it is that routine's responsibility to deal with it according to its needs. If the data were numeric, as with a menu selection, then the value of TT$ is taken. If string data is desired, the string you wish to contain the data is set equal to TT$. This routine will only return to the calling routine when the ENTER key is pressed.



Line 1020 is the beginning of the routine that prints the logo and copyright notice at the beginning and the end of the program. Line 1200 starts the routine to print the instructions on the screen. When you key this program in, it would be advisable to type in all spaces and characters exactly as shown to preserve the tight screen formatting I have used. After the program is up and running, you can modify it. It is much easier to modify a program that works than one that doesn't. The main menu is printed in lines 1400 to 1510. The error trap in line 1690 will always return you to the main menu if any error occurs. For this reason, it is recommended that you not key in line 110 (the ONERRORGOTO statement) until you have checked for syntax and other typographical errors.

The sort routine begins in line 1530. The word SORT is printed in large graphic characters on the screen to inform the user that the sort is in progress. The values of X and Y are displayed on the screen at all times along with the the value being sorted on the left and the item being compared to it on the right. If the item on the right is less than the item on the left, the computer will exchange the two values, thus accomplishing the sort. If the program seems to hang while sorting a large file, give it some time before hitting the reset button, because the computer might be in a garbage collection procedure and will resume normal operations in a couple of minutes without operator intervention.

This program was written with the needs of the small business in mind. It emphasizes ease of data entry over flexibility; after all, the salary expense for a secretary in a new business is a more significant portion of available capital than in an older establishment. These data files will not grow out of date as with other types of mailing lists, because if the residence has not burned down or been otherwise destroyed, it still exists. Today's vacancy rates are low, so there should be a high rate of return on advertising sent out in this manner. This program will be a great asset to any small business owning a TRS-80.

Reference	Line Number
00130	160, 1690
00140	140
00170	150
00200	200
00230	230
00250	210
00260	330
00290	310
00340	150, 310, 320
00360	360
00450	420
00480	150
00500	500
00560	600
00610	150
00720	660
00730	150
00750	750
00760	840
00820	850
00860	170, 340, 480, 610, 730
00890	200, 230, 360, 500, 630
00900	140, 310, 610, 1110
00910	910, 930

Table continued

00940	920
00950	920
00980	920
01020	120, 1520
01130	250
01200	1110
01370	1300
01380	1380
01390	130
01520	150
01530	150
01660	1640
01690	110
A	100, 140, 150
A$	820, 830, 840
B$	1650
C	900, 940
CZ$	220, 260, 300, 380, 530, 700
E	140, 200, 230, 310, 360, 500, 610, 630, 900, 960, 1110
FS$	360, 370, 500, 520
I	260, 270, 280, 310, 320, 340, 410, 550, 560, 570, 580, 590, 1610, 1620, 1630
M1	900, 910, 940, 960, 980, 990, 1000, 1010
NA$	100, 260, 300, 630, 680
NU$(	90, 100, 260, 310, 320, 420, 430, 440, 580 590, 660, 690, 1630, 1640, 1650
S	140, 200, 230, 310, 360, 500, 610, 630, 900, 980, 990, 1000, 1010, 1110
ST$	240, 260, 300, 390, 540, 690
TI	340, 570, 650
TT$ $	140, 200, 210, 220, 230, 240, 310, 360, 500, 630 900, 940, 960, 990, 1010, 1110
WL	470, 570, 640, 1380, 1690
WL%	910
X	410, 420, 430, 440, 450, 650, 660, 690, 720, 1610, 1620, 1630, 1640, 1650, 1670
X$	900, 910, 920, 950, 960
Y	1620, 1630, 1640, 1650, 1660
YZ	1090, 1520
Z	100

Table 1. *Variables and references in Maillist*

Program Listing

Encyclopedia Loader™

```
 10 REM          *************************************************
 20 REM          **       SIMPLE OCCUPANT MAILING LIST          **
 30 REM          ** COPYRIGHT (c) 1981 BY CHARLES P. KNIGHT     **
 40 REM          **       MINIMUM SYSTEM: TRS-80 32K            **
 50 REM          **         ONE OR MORE DISK DRIVES             **
 60 REM          **               03/01/81                      **
 70 REM          *************************************************
 80 CLEAR 50:
    CLEAR 2 * MEM / 3
 90 DIM NU$(1500)
100 DEFINT A - Z:
    NA$ = "OCCUPANT":
    NU$(0) = "XXXXX"
110 ON ERROR GOTO 1690
120 GOSUB 1020
130 GOSUB 1390
140 S = 863:
    E = 864:
    GOSUB 900:
    A = VAL(TT$):
    IF A > 7 OR A < 1
     THEN
      140
150 ON A GOSUB 170,480,610,730,1530,1520,340
160 GOTO 130
170 GOSUB 860
180 PRINT @452,"ENTER CITY, STATE, ZIP :";
190 PRINT @516,"NAME OF STREET :";
200 S = 477:
    E = 505:
    GOSUB 890:
    IF TT$ = ""
     THEN
      200
210 IF TT$ = "ADD-ON"
     THEN
      250
220 CZ$ = TT$
230 S = 533:
    E = 555:
    GOSUB 890:
    IF TT$ = ""
     THEN
      230
240 ST$ = " " + TT$
250 GOSUB 1130
260 PRINT @64,NA$;:
    PRINT @128,NU$(I)ST$;"    ";:
    PRINT @192,CZ$;
270 I = I + 1
280 PRINT @115,I;:
    PRINT @179,1500 - I;
290 PRINT @333,"ENTER NEXT ADDRESS OR " CHR$(34)"@" CHR$(34)" TO STO
    P ENTRY";
300 PRINT @596,NA$;:
    PRINT @667,ST$;:
    PRINT @724,CZ$;
310 S = 660:
    E = 666:
    GOSUB 900:
    IF TT$ = ""
     THEN
      290:
     ELSE
      IF TT$ = "@"
       THEN
        340:
```

Program continued

```
            :
          ELSE
           NU$(I) = TT$
320 IF NU$(I) = "@"
     THEN
      340
330 GOTO 260
340 TI = I - 1:
    GOSUB 860
350 PRINT @452,"ENTER FILESPEC FOR THIS FILE:";
360 S = 584:
    E = S + 15:
    GOSUB 890:
    IF TT$ = ""
     THEN
      360:
     ELSE
      FS$ = TT$
370 OPEN "O",1,FS$
380 PRINT #1,CZ$
390 PRINT #1,ST$
400 PRINT @0,"WRITING RECORD #";
410 FOR X = 1 TO I
420  IF NU$(X) = "" OR NU$(X) = "@"
      THEN
       450
430  PRINT #1,NU$(X);",";
440  PRINT @16,X"   "NU$(X)"     ";
450  NEXT X
460 CLOSE
470 FOR WL = 1 TO 1000:
     NEXT WL:
    RETURN
480 GOSUB 860
490 PRINT @452,"ENTER FILESPEC TO BE LOADED:";
500 S = 584:
    E = S + 15:
    GOSUB 890:
    IF TT$ = ""
     THEN
      500:
     ELSE
      FS$ = TT$
510 PRINT @0,"LOADING FILE #";
520 OPEN "I",1,FS$
530 LINE INPUT #1,CZ$
540 LINE INPUT #1,ST$
550 I = 0
560 I = I + 1
570 IF EOF (1)
     THEN
      CLOSE :
      TI = I - 1:
      FOR WL = 1 TO 1000:
       NEXT WL:
      RETURN
580 INPUT #1,NU$(I)
590 PRINT @14,I"   "NU$(I)"     ";
600 GOTO 560
610 GOSUB 860:
    PRINT @452,"ALIGN LABELS IN PRINTER, THEN PRESS <ENTER>";:
    S = 496:
    E = 497:
    GOSUB 900
620 PRINT @452, CHR$(249);:
    PRINT @452,"ENTER THE MESSAGE FOR THE FIRST LINE OF THE LABEL";
630 S = 517:
    E = S + 32:
    GOSUB 890:
    IF TT$ = ""
```

```
     THEN
      NA$ = "OCCUPANT":
     ELSE
      NA$ = TT$
640 FOR WL = 452 TO 644 STEP 64:
     PRINT @WL, CHR$(249);:
     NEXT WL
650 FOR X = 1 TO TI
660  IF NU$(X) = "@" OR NU$(X) = ""
      THEN
       720
670  IF INKEY$ = "@"
      THEN
       RETURN
680  LPRINT "":
     LPRINT NA$:
     PRINT @452,NA$;
690  LPRINT NU$(X);ST$:
     PRINT @516,NU$(X)ST$"      ";
700  LPRINT CZ$:
     PRINT @580,CZ$;
710  LPRINT
720  NEXT X:
    RETURN
730 GOSUB 860
740 PRINT @452,"ALIGN LABLES THEN PRESS <ENTER>";
750 IF INKEY$ < > CHR$(13)
     THEN
      750
760 LPRINT
770 LPRINT STRING$(24,88)
780 LPRINT STRING$(24,88)
790 LPRINT STRING$(24,88)
800 LPRINT
810 PRINT @516,"PRESS ENTER TO ALIGN AGAIN, SPACE BAR TO RETURN";
820 A$ = INKEY$
830 IF A$ = CHR$(32)
     THEN
      RETURN
840 IF A$ = CHR$(13)
     THEN
      760
850 GOTO 820
860 CLS
870 PRINT @320, STRING$(67,191);:
    PRINT @445, STRING$(6,191);:
    PRINT @509, STRING$(6,191);:
    PRINT @573, STRING$(6,191);:
    PRINT @637, STRING$(6,191);:
    PRINT @701, STRING$(3,191);:
    PRINT @704, STRING$(64,191);
880 RETURN
890 REM           DATA INPUT SUBROUTINE
900 C = E + 1 - S:
    TT$ = "":
    PRINT @S, STRING$(C,".");:
    M1 = S:
    X$ = INKEY$
910 FOR WL% = 1 TO 5:
     NEXT WL%:
    PRINT @M1, CHR$(140);:
    X$ = INKEY$:
    FOR WL% = 1 TO 5:
     NEXT WL%:
    PRINT @M1,".";:
    IF X$ = ""
     THEN
      910
920 IF X$ = CHR$(13)
     THEN
```

Program continued

```
        940:
       ELSE
        IF X$ = CHR$(8)
         THEN
          GOSUB 980:
         ELSE
          GOSUB 950
 930 GOTO 910
 940 TT$ = LEFT$(TT$,C):
     PRINT @M1, STRING$(C - LEN(TT$),32);:
     RETURN
 950 IF ASC(X$) < 32 OR ASC(X$) > 122
      THEN
       RETURN
 960 PRINT @M1,X$;:
     TT$ = TT$ + X$:
     M1 = M1 + 1:
     IF M1 > E
      THEN
       M1 = E
 970 RETURN
 980 M1 = M1 - 1:
     IF M1 < S
      THEN
       M1 = S
 990 PRINT @M1,".";:
     TT$ = LEFT$(TT$,M1 - S)
1000 IF M1 < S
      THEN
       M1 = S
1010 TT$ = LEFT$(TT$,M1 - S):
     RETURN
1020 CLS :
     REM  LOGO PRINTING ROUTINE
1030 PRINT @141, STRING$(35,191);:
     PRINT @205, CHR$(191) STRING$(33,143) CHR$(191);
1040 PRINT @269, CHR$(191);"  OCCUPANT MAILING LIST PROGRAM  ";
     CHR$(191);
1050 PRINT @333, STRING$(6,191)" COPYRIGHT (c) 1981 BY " STRING$(6,19
     1);
1060 PRINT @397, STRING$(8,191)" CHARLES P. KNIGHT " STRING$(8,191);
1070 PRINT @461, STRING$(12,191)" ALL RIGHTS " STRING$(11,191);
1080 PRINT @525, STRING$(13,191)" RESERVED " STRING$(12,191);:
     PRINT @589, STRING$(35,191);
1090 IF YZ
      THEN
       RETURN
1100 PRINT @845,"DO YOU NEED INSTRUCTIONS?";
1110 S = 872:
     E = 875:
     GOSUB 900:
     IF TT$ = "Y"
      THEN
       GOSUB 1200
1120 RETURN
1130 CLS :
     REM  DATA INPUT SCREEN
1140 PRINT @0,"LAST ADDRESS ENTERED :";:
     PRINT @34, CHR$(191) STRING$(28,131) CHR$(191);
1150 PRINT @98, CHR$(191);" ENTRY NUMBER :";:
     PRINT @127, CHR$(191);
1160 PRINT @162, CHR$(191);" ENTRIES LEFT :";:
     PRINT @191, CHR$(191);:
     PRINT @226, CHR$(143) STRING$(28,140) CHR$(143);:
     PRINT @384, CHR$(191) STRING$(62,131);
1170 PRINT @447, STRING$(2,191);:
     PRINT @511, STRING$(2,191);:
     PRINT @575, STRING$(2,191);:
     PRINT @639, STRING$(2,191);:
     PRINT @703, STRING$(2,191);:
```

```
     PRINT @767, STRING$(2,191);:
     PRINT @831, STRING$(2,191) STRING$(62,176);:
     PRINT @895, CHR$(191);
1180 PRINT @969,"PROGRAM MAY PAUSE OCCASIONALLY TO PROCESS DATA";
1190 RETURN
1200 CLS
1210 PRINT @199, STRING$(45,191);:
     PRINT @263, CHR$(191);:
     PRINT @307, CHR$(191);:
     PRINT @327, CHR$(191);:
     PRINT @371, CHR$(191);:
     PRINT @391, CHR$(191);:
     PRINT @435, CHR$(191);:
     PRINT @455, CHR$(191);:
     PRINT @499, CHR$(191);
1220 PRINT @519, CHR$(191);:
     PRINT @563, CHR$(191);:
     PRINT @583, CHR$(191);:
     PRINT @627, CHR$(191);:
     PRINT @647, CHR$(191);:
     PRINT @691, CHR$(191);
1230 PRINT @711, STRING$(45,191);
1240 PRINT @330,"THIS  PROGRAM  IS  A  SIMPLIFIED  ENTRY";
1250 PRINT @394,"OCCUPANT  MAILING  LIST SYSTEM.  IN THE";
1260 PRINT @458,"ENTRY  PHASE  OF THE PROGRAM,  YOU NEED";
1270 PRINT @522,"ONLY  FILL  IN THE  BLANKS.  THE  HOUSE";
1280 PRINT @586,"NUMBERS  TO BE ENTERED  MUST  ALL BE ON";
1290 PRINT @650,"THE SAME STREET  AND HAVE THE SAME ZIP.";
1300 GOSUB 1370
1310 PRINT @329,"YOU WILL CREATE  A SEPARATE  NAMED  DISK";
1320 PRINT @393,"FILE  FOR EVERY  CHANGE  OF STREET  NAME";
1330 PRINT @457,"CITY,  STATE, OR ZIP CODE.  THIS  ALLOWS";
1340 PRINT @521,"PREVIOUSLY  UNPRECEDENTED  EASE  OF DATA";
1350 PRINT @585,"ENTRY.  BE SURE  TO SAVE  THE FILE USING";
1360 PRINT @649,"OPTION  2 BEFORE  EXITING  THE  PROGRAM.";
1370 PRINT @903,"PRESS <     > TO CONTINUE...................";
1380 PRINT @910,"ENTER";:
     FOR WL = 1 TO 40:
      NEXT WL:
     PRINT @910, STRING$(5,32);:
     FOR WL = 1 TO 32:
      NEXT WL:
     IF INKEY$ < > CHR$(13)
      THEN
       1380:
      ELSE
       RETURN
1390 CLS
1400 PRINT @8,"MAILING LIST MASTER MENU. ENTER SELECTION BELOW";
1410 PRINT @128, STRING$(64,191);
1420 PRINT @272,"<1>   ENTER DATA FOR A NEW STREET";
1430 PRINT @336,"<2>   INPUT DATA FROM DISK";
1440 PRINT @400,"<3>   PRINT MAILING LABELS";
1450 PRINT @464,"<4>   ALIGN LABELS ON PRINTER";
1460 PRINT @528,"<5>   SORT HOUSE NUMBERS";
1470 PRINT @592,"<6>   END PROGRAM";
1480 PRINT @656,"<7>   WRITE FILE TO DISK";
1490 PRINT @704, STRING$(64,191);
1500 PRINT @854,"SELECT :";
1510 RETURN
1520 YZ = - 1:
     GOSUB 1020:
     CLOSE :
     CLEAR 50:
     END
1530 CLS :
     PRINT @192, STRING$(12,191) STRING$(4,32) STRING$(8,191)
     STRING$(7,191) STRING$(3,32) STRING$(14,191)" " STRING$(16,191);
1540 PRINT @272, STRING$(2,191);:
     PRINT @285, STRING$(2,191) STRING$(3,32) STRING$(2,191);:
```

Program continued

```
     PRINT @302, STRING$(2,191);
1550 PRINT @312, STRING$(2,191);:
     PRINT @320, CHR$(191);:
     PRINT @336, STRING$(2,191);:
     PRINT @349, STRING$(2,191) STRING$(3,32) STRING$(2,191);:
     PRINT @366, STRING$(2,191);
1560 PRINT @376, STRING$(2,191);:
     PRINT @384, STRING$(12,191) STRING$(4,32) STRING$(2,191);:
     PRINT @413, STRING$(2,191) STRING$(3,32) STRING$(14,191);:
     PRINT @440, STRING$(2,191);
1570 PRINT @459, CHR$(191) STRING$(4,32) STRING$(2,191);:
     PRINT @477, STRING$(2,191) STRING$(3,32) STRING$(2,191)
     STRING$(6,32) STRING$(2,191);:
     PRINT @504, STRING$(2,191);:
     PRINT @523, STRING$(2,191);:
     PRINT @523, CHR$(191) STRING$(4,32) STRING$(2,191);
1580 PRINT @504, STRING$(2,191);:
     PRINT @523, CHR$(191) STRING$(4,32) STRING$(2,191);:
     PRINT @541, STRING$(2,191) STRING$(3,32) STRING$(2,191)
     STRING$(6,32) STRING$(2,191);:
     PRINT @568, STRING$(2,191);
1590 PRINT @576, STRING$(12,191) STRING$(4,32) STRING$(8,191)
     STRING$(7,191) STRING$(3,32) STRING$(2,191) STRING$(6,32)
     STRING$(2,191) STRING$(3,191) STRING$(9,32) STRING$(2,191);
1600 PRINT @832,"SORTING PASS :";:
     PRINT @870,"SORTING PASS :";:
     PRINT @912,"MAKING";:
     PRINT @927,"PASSES";
1610 FOR X = 1 TO I - 1
1620  FOR Y = X + 1 TO I
1630   PRINT @714,NU$(X)"     ";:
       PRINT @742,NU$(Y)"     ";:
       PRINT @885,X" ";:
       PRINT @847,Y" ";:
       PRINT @920,I - 1;
1640   IF VAL(NU$(X)) < = VAL(NU$(Y))
        THEN
         1660
1650   B$ = NU$(X):
       NU$(X) = NU$(Y):
       NU$(Y) = B$
1660   NEXT Y
1670  NEXT X
1680 RETURN
1690 CLS :
     PRINT @512,"AN ERROR HAS OCCURRED. RETURNING TO MENU.":
     PRINT "ERROR : " ERR / 2 + 1"  IN LINE # " ERL :
     FOR WL = 1 TO 2500:
      NEXT WL:
     RESUME 130
```

Business Forms— The Statement

by R. L. Conhaim

A statement is a very useful business form. It indicates to the customer exactly what is owed and what has been paid since the last report, and gives the customer a chance to check his or her own records against the supplier's records.

The form of the statement depends on the type of business using it. As used in most businesses, the statement is a compilation of open invoices and a record of payments. In some cases, however, as with doctors and dentists, the statement is a record of visits, charges, and payments. When this is the case, the statement is the only record the patient receives. Though the patient doesn't receive an invoice, charges are made on the books just as though invoices had been issued. Since invoices and charges

STATEMENT

FROM:	DATE: 7/1/81
YOUR COMPANY NAME	ACCOUNT NO. 1234
YOUR COMPANY STREET ADDRESS	
CITY, STATE, ZIP	

TO:

BECHHOLDT CONSTRUCTION CO.
6598 RIVERSIDE RD
VICTORVILLE

THIS STATEMENT REFLECTS PAYMENTS
RECEIVED THRU 6/25/81. IF YOUR
RECORDS DO NOT AGREE WITH OURS
CALL NANCY SMITH AT 227-8657.

OPENING BALANCE 56.89

INVOICE NO.	DATE	YOUR P.O. NO.	CHARGE	PAYMENT	BALANCE
5678	6/18/81	B6784	121.34		178.23
				56.89	121.34
5781	6/21/81	B6799	107.77		229.11
5799	6/23/81	B6807	54.29		283.40
		LAST ITEM			

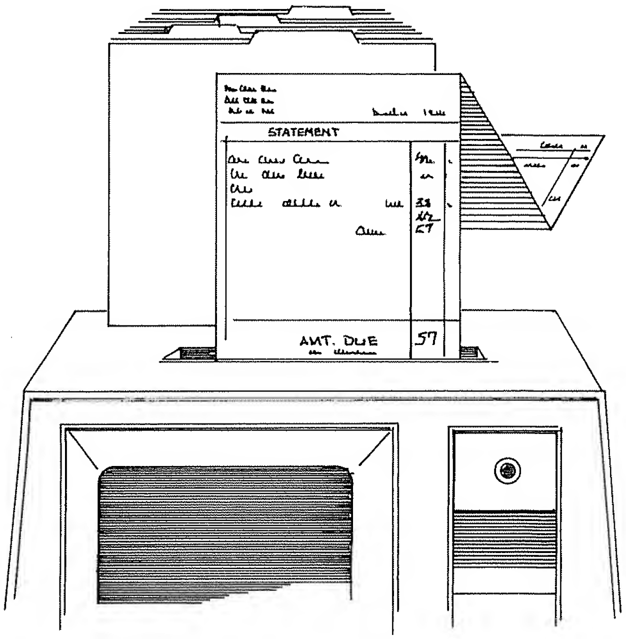
PAY LAST AMOUNT IN BALANCE COLUMN

Figure 1. *The statement*

have been entered into the bookkeeping system by the time the statement is issued, it is not a primary record. It is, however, useful for the customer in reconciling his or her records. It is also a kind of low-key reminder that invoices due have not been paid and can be an excellent form of public relations when it carries a friendly tone.

The statement shown in Figure 1 contains more information than is normally used in a statement, because it has been designed to give the customer as much information as possible to check his or her records thoroughly. A name and telephone number to call, if there is a discrepancy in the records, is included for the customer's benefit. The program for the statement is quite straightforward. You would, of course, put your own name and address in lines 130 through 150, and the name of your bookkeeper and telephone number in line 300.

The INKEY function is used to reduce the number of key pressings where the program asks if you have another entry or want to make another statement. Because the statement is not a primary record and need not be interactive with the bookkeeping system, the program does not provide for recording the statement on disk or cassette. The statement has been designed to mail in a number 10 (business size) window envelope. If your printer provides for reduced size print, the statement could also be made to fit in a smaller envelope.



Program Listing. *Statement program*

Encyclopedia Loader™

```
10 'XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
20 'STATEMENT PRINT PROGRAM
30 'BY R. L. CONHAIM
40 '15506 KIAMICHI ROAD, APPLE VALLEY, CA 92307
50 'XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
60 CLS
70 CLEAR 1000
80 INPUT "ENTER CUSTOMER'S ACCOUNT NUMBER";AN
90 INPUT "ENTER DATE";D1$
100 INPUT "ENTER STATEMENT CLOSING DATE";D2$
110 LPRINT TAB(29) CHR$(01);"STATEMENT"; CHR$(02):
    LPRINT
120 LPRINT "FROM:"; TAB(46)"DATE:";D1$
130 LPRINT "YOUR COMPANY NAME"; TAB(46) "ACCOUNT NO.";AN
140 LPRINT "YOUR COMPANY STREET ADDRESS"
150 LPRINT "CITY, STATE, ZIP"
160 LPRINT :
    LPRINT
170 LPRINT STRING$(75,"-")
180 INPUT "CUSTOMER'S NAME";N$
190 INPUT "CUSTOMER'S STREET ADDRESS";A$
200 INPUT "CITY, STATE, ZIP";C$
210 INPUT "ENTER OPENING BALANCE";BA
220 LPRINT "TO:":
    LPRINT
230 LPRINT TAB(15) N$
240 LPRINT TAB(15) A$
250 LPRINT TAB(15) C$
260 LPRINT :
    LPRINT :
    LPRINT
270 LPRINT TAB(25) "THIS STATEMENT REFLECTS PAYMENTS"
280 LPRINT TAB(25) "RECEIVED THRU ";D2$;".IF YOUR "
290 LPRINT TAB(25) "RECORDS DO NOT AGREE WITH OURS"
300 LPRINT TAB(25) "CALL NANCY SMITH AT 227-8657."
310 LPRINT :
    LPRINT
320 M$ = "##,###.##"
330 LPRINT TAB(46)"OPENING BALANCE"; TAB(61) USING M$;BA
340 LPRINT STRING$(75,"-"):
    LPRINT
350 LPRINT "INVOICE NO."; TAB(19)"DATE"; TAB(31)"YOUR P.O.NO.";
    TAB(47)"CHARGE"; TAB(55)"PAYMENT"; TAB(63)"BALANCE"
360 LPRINT STRING$(75,"-"):
    LPRINT
370 PRINT "PRESS C FOR CHARGE, P FOR PAYMENT"
380 DEFSTR T
390 T = INKEY$:
    IF T = ""
     THEN
      390
400 IF T = "C" GOTO 430
410 IF T = "P" GOTO 520
420 GOTO 370
430 INPUT "INVOICE NUMBER";I$
440 INPUT "INVOICE DATE";D3$
450 INPUT "CUSTOMER'S P.O.NUMBER";P$
460 INPUT "ENTER AMOUNT OF CHARGE";CH
470 M$ = "##,###.##"
480 BA = BA + CH
490 LPRINT I$; TAB(17)D3$; TAB(35)P$;
500 LPRINT TAB(43) USING M$;CH;
510 GOTO 550
520 INPUT "ENTER AMOUNT OF PAYMENT";PMT
530 LPRINT TAB(51) USING M$;PMT;
540 BA = BA - PMT
550 LPRINT TAB(61) USING M$;BA
560 PRINT "ANOTHER ENTRY? Y OR N"
```

Program continued

```
570  T = INKEY$:
     IF T = ""
      THEN
       570
580  IF T = "Y" GOTO 370
590  IF T = "N" GOTO 610
600  GOTO 560
610  LPRINT TAB(22)"LAST ITEM"
620  LPRINT STRING$(75,"-")
630  LPRINT
640  LPRINT TAB(32) "PAY LAST AMOUNT IN BALANCE COLUMN"
650  CH = 0:
     PMT = 0:
     :
     BA = 0
660  LPRINT CHR$(12)
670  PRINT "ANOTHER STATEMENT? Y OR N"
680  T = INKEY$:
     IF T = ""
      THEN
       680
690  IF T = "Y" GOTO 80
700  IF T = "N" GOTO 720
710  GOTO 670
720  END
```

EDUCATION

Grade Calculator
Classroom Doodles

Grade Calculator

by Robert C. Jacobs, Ph.D.

As a teacher, I am often faced with having to calculate grades for large numbers of students. The relative weights of the exercises that determine the grade vary considerably, making the job of computing averages a time-consuming one. This seems an ideal problem for a computer; the work is repetitive, tedious, and all too frequent. Where there are many classes to prepare grades for, this program saves much drudgery. It also offers the advantage of versatility. While the program uses my grading technique, it will adapt to most other grading systems with only minor changes.

I assign numerical marks to students on a scale of fifty to one hundred, so that 85 and 94 express B and A respectively. Table 1 gives the precise range of scores for each letter grade. At the end of the academic period, the numerical grades are weighted, and an average computed. The grade weight for each exercise is usually set and announced to the class at the beginning of classes. For example, last winter I assigned a paper (33%), two examinations (25% each), and gave a short quiz (17%). Though this may sound complicated, the computer makes the calculation of a student's final average easy. After the average is determined, the program assigns a letter grade. But, the instructor is given the opportunity to change the grade in the light of any special circumstances which may exist for a particular student. After all the students' grades are figured, the computer tallies the grades for a class grade point average.

Your first entry indicates how many scores or marks are to be entered for each student. Then in lines 130 to 150 you're asked to enter the weight of each score. The weights are expressed in percentages; no decimal point is used. As the program is written, the total of these must be 100. After the percentages are entered, the students' scores are entered. These are also expressed in percentages, because the arithmetic in line 210 requires it. If 999 is entered as the score, the program breaks out of its loop and goes to the final class report.

After the entry of each student's scores, the program repeats the scores and shows the student's average and letter grade. At this point, you may alter the grade if you wish. In lines 430 to 520 a tally is kept of the grades and the total grade points, so that the class G.P.A. may be computed at the end. These lines can be altered if your school assigns different grade points to the letter grades. Revising this sytem is fairly simple. The key elements of this program are found in lines 280 to 380[1] and 430 to 510[2], respectively.

These establish the relationship between score percentage and letter grade and the relationship between letter grade and grade points. These values are listed in Table 1. A list of variables is provided in Table 2 for convenience in revising.

Lines 230, 240, 410, and 420 allow the program to respond to upper- and lowercase input without disturbing the rhythm of your work. To enter the lowercase *y* or *n* in the quote strings, just hold the shift key down if you are using an unmodified Level II machine or a Model III with the lowercase switched off. When entering line 490, be sure you put a minus sign in the quotation marks.

I hope you will find this program as useful as I've found it. It is a great time-saver that bridges the gap between fully computerized student record-keeping and the paper-and-pencil method of grade computation.

Grade in percent	Letter grade	Grade points
50–59[3]	F	0.0
60–62	D–	0.7
63–66	D	1.0
67–69	D+	1.3
70–72	C–	1.7
73–76	C	2.0
77–79	C+	2.3
80–82	B–	2.7
83–86	B	3.0
87–89	B+	3.3
90–92	A–	3.7
93–100[3]	A	4.0

Table 1. *Grade relationships*

AV	Increments of weighted scores
TP	Total grade points
N	Number of students: (N–1) = number of students
S	Number of scores comprising each student's average
G(X)	Scores entered
W(X)	Weights
X	General counter
A$	Answer string for INKEY$ function
G1–G5	Count As, Bs, and Fs respectively
TW	Accumulates total of the weights
GP	Accumulates grade points: GP/N = G.P.A.

Table 2. *Variable list*

[1]The relationships between percentage and letter grade are in lines 280–380.

[2]The relationships between letter grade and grade points may be found in lines 430–470. Lines 490–510 calculate grade points for + and – grades.

[3]At my school, the grade F does not take plus or minus, nor is there an A+.

Program Listing

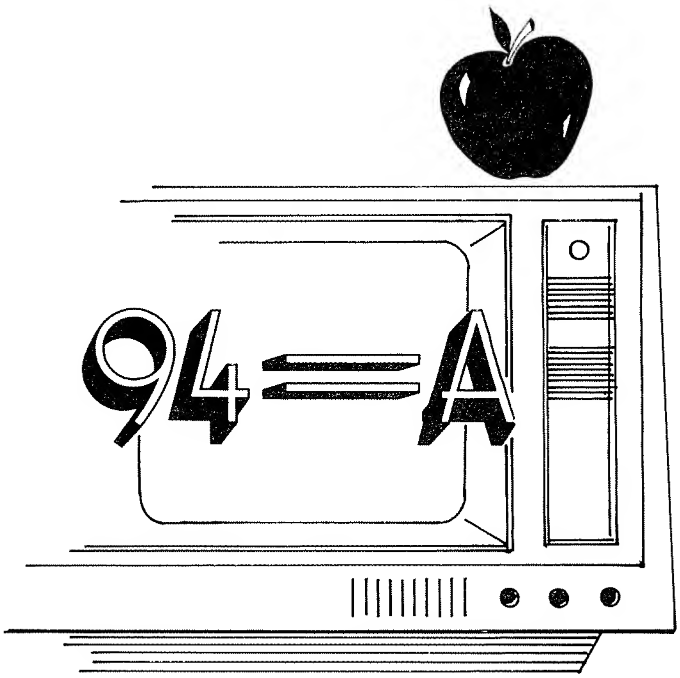
Encyclopedia Loader

```
 10 CLS :
    PRINT TAB(24)"GRADE CALCULATOR":
    PRINT
 20 FOR X = 48 TO 80:
     SET(X,3):
     NEXT
 30 PRINT TAB(30),"by"
 40 PRINT TAB(25)"Robert C. Jacobs"
 50 PRINT TAB(27)"April, 1981":
    PRINT
 60 PRINT "   This program will figure averages and letter grades"
 70 PRINT "of students and has the capacity to calculate the grade p
    oint"
 80 PRINT "average for an entire class."
 90 PRINT "   The instructor should enter the number of grades which
     will"
100 PRINT "figure in the average and then the weight to be given to
    each."
110 PRINT "Weights should be expressed in percentages, and must add
    to 100%"
120 INPUT "How many test scores or grades will you use";S:
    DIM W(S),G(S)
130 CLS :
    PRINT "Weight for each score or test grade;";S;"weights are requ
    ired.":
    PRINT :
    TW = 0
140 FOR X = 1 TO S:
     PRINT "What percentage for score";X;:
     INPUT W(X):
     TW = TW + W(X):
     NEXT
150 IF INT(TW + .1) < > 100
     THEN
      CLS :
      PRINT @407,"TOTAL IS NOT 100%":
      FOR X = 0 TO 800:
       NEXT :
      GOTO 130
160 CLS :
    PRINT "ENTER 999 TO END AND GET CLASS REPORT":
    PRINT
170 N = N + 1
180 AV = 0:
    FOR X = 1 TO S
190  PRINT "Test score";X;"for student";N;:
     INPUT G(X):
     IF G(X) = 999
      THEN
       540
200  IF G(X) > 100 OR G(X) < 0
      THEN
       190
210  AV = AV + G(X) * W(X) / 100:
     PRINT :
     NEXT
220 PRINT "Are these entries correct (Y/N)?":
    GOSUB 650
230 IF (A$ = "N") OR (A$ = "n") GOTO 180
240 IF (A$ < > "Y") AND (A$ < > "y")
     THEN
      220
250 CLS :
    PRINT "This student´s marks were:":
    PRINT
260 FOR X = 1 TO S:
     PRINT "Score";X;": ";G(X),:
```

Program continued

```
     NEXT
270 PRINT :
    PRINT :
    PRINT "The average is: ";:
    PRINT USING "###.#%";AV
280 IF AV < 60
     THEN
      G$ = "F":
      GOTO 390
290 IF AV < 63
     THEN
      G$ = "D-":
      GOTO 390
300 IF AV < 67
     THEN
      G$ = "D":
      GOTO 390
310 IF AV < 70
     THEN
      G$ = "D+":
      GOTO 390
320 IF AV < 73
     THEN
      G$ = "C-":
      GOTO 390
330 IF AV < 77
     THEN
      G$ = "C":
      GOTO 390
340 IF AV < 80
     THEN
      G$ = "C+":
      GOTO 390
350 IF AV < 83
     THEN
      G$ = "B-":
      GOTO 390
360 IF AV < 87
     THEN
      G$ = "B":
      GOTO 390
370 IF AV < 90
     THEN
      G$ = "B+":
      GOTO 390
380 IF AV < 93
     THEN
      G$ = "A-":
     ELSE
      G$ = "A"
390 PRINT :
    PRINT "Based on this student´s average, the grade is ";G$
400 PRINT :
    PRINT "Substitute a different grade (Y/N)?":
    GOSUB 650
410 IF (A$ = "Y") OR (A$ = "y")
     THEN
      INPUT "Enter correct grade and then press <ENTER>";G$:
      GOTO 430
420 IF (A$ < > "N") AND (A$ < > "n")
     THEN
      400
430 IF LEFT$(G$,1) = "A"
     THEN
      GP = 4:
      G1 = G1 + 1:
      GOTO 480
440 IF LEFT$(G$,1) = "B"
     THEN
      GP = 3:
```

```
      G2 = G2 + 1:
      GOTO 480
450 IF LEFT$(G$,1) = "C"
     THEN
      GP = 2:
      G3 = G3 + 1:
      GOTO 480
460 IF LEFT$(G$,1) = "D"
     THEN
```



```
      GP = 1:
      G4 = G4 + 1:
      GOTO 480
470 IF LEFT$(G$,1) = "F"
     THEN
      GP = 0:
      G5 = G5 + 1:
      GOTO 480
480 IF GP = 0 GOTO 520
490 IF RIGHT$(G$,1) = "-"
     THEN
      GP = GP - .3
500 IF GP = 4 GOTO 520
510 IF RIGHT$(G$,1) = "+"
     THEN
      GP = GP + .3
520 TP = TP + GP:
530 GOTO 160
540 CLS :
    PRINT TAB(22)"C L A S S   R E P O R T"
550 FOR X = 44 TO 89:
     SET(X,3):
     NEXT
560 PRINT :
```

Program continued

```
    PRINT TAB(14)"THE CLASS GRADE POINT AVERAGE IS ";:
    PRINT USING "#.##";TP / (N - 1)
570 PRINT :
    PRINT TAB(13)"GRADE",,"NUMBER"
580 PRINT TAB(14)"A",,,G1
590 PRINT TAB(14)"B",,,G2
600 PRINT TAB(14)"C",,,G3
610 PRINT TAB(14)"D",,,G4
620 PRINT TAB(14)"F",,,G5
630 PRINT TAB(47)"-----":
    PRINT TAB(48)N - 1
640 END
650 A$ = INKEY$:
    IF A$ = ""
     THEN
      650:
     ELSE
      RETURN
```

Classroom Doodles

by Ann Rosenberg

Graphics in the classroom not only teach computer math but also promote problem-solving, creativity, and mental curiosity.

Stimulating a high school computer math class the week before the spring vacation was my main objective in designing this project. The students had been in the course for only eight weeks. We had studied system commands such as NEW, LIST, DELETE, CSAVE, CLOAD, and EDIT; and program statements such as INPUT, FOR-NEXT, READ-DATA, INT, RND, IF-THEN-ELSE, GOSUB, and ON N GOTO. The group had successfully written several math-oriented programs, but a change was now in order. The class needed a project which was educational and fun. After rejecting several ideas, we decided on a graphics assignment.

Using Drawings

Each of the 12 students was given the following assignment: Using the TRS-80 Video Worksheet, draw a picture using horizontal, vertical, and diagonal lines. From this, use SET(X,Y) and RESET(X,Y) to write the corresponding coding. After writing the program, they were asked to type it, debug it, and place the completed program on tape.

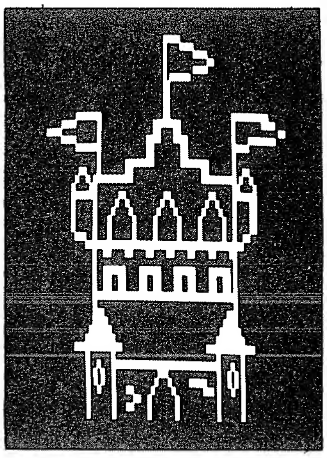
Their first reaction was "What should I draw?" Until this assignment, the students had been given exact instructions on what their programs were to do and how. Now they seemed at a loss, but this changed quickly as they put their imaginations to work.

Before long, several programs were written and put to the test on the computer. After looking at their graphics results, most of the students weren't satisfied with their simple, stationary drawings. They went back to their worksheets to create more sophisticated ones.

The following are examples of what the students developed:

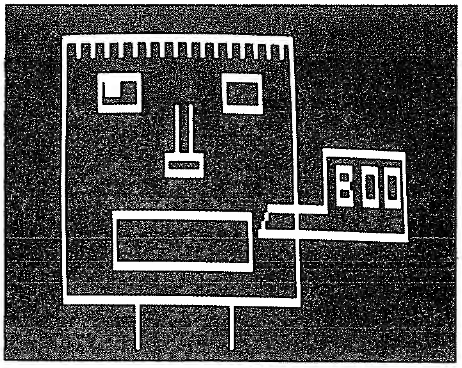
- Program Listing 1—Started out as a simple house, but ended up a castle.
- Program Listing 2—Was a strange face, but became a Frankenstein, complete with moving lips and shifting eyes.
- Program Listing 3—Was an Easter Bunny, but it was quickly transformed into a Playboy Bunny with blinking eyes.
- Program Listing 4—Was a plain boat until the letters USA were added and made to appear as though they moved across the body of the ship.

● Program Listing 5—Is the class favorite. The first day it was a simple stop sign. The second day, the student added a dog. With a little prodding from classmates, the student had the dog add a few "plops" at the base of the sign.
● Program Listing 6—Is the advertising logo for a lumber company owned by a student's father.



By the end of the week, everyone had completed exciting and creative graphics displays. Each was so proud of his or her accomplishments that it was not unusual for friends and teachers to stop by the computer room and view the drawings.

The students not only enjoyed this assignment, but they became proficient at using graphics. Future assignments will be much easier for them. Instead of just solving right triangles with the Pythagorean theorem or general triangles with the law of sines and cosines, they will be able to draw these triangles. The graphics can also be used to make bar graphs, display data, and write games.



Program Listing 1. *Castle*

Encyclopedia Loader

```
  1 REM    ******** CASTLE ********
  2 REM  PROGRAMMER: MICHAEL SHLENKER (SOPHOMORE)
  5 CLS
 10 FOR A = 35 TO 79:
     SET(A,27):
     NEXT A
 20 FOR B = 37 TO 77:
     SET(B,33):
     NEXT B
 30 FOR C = 32 TO 44 :
     SET(C,39):
     NEXT C
 40 FOR D = 69 TO 82:
     SET(D,39):
     NEXT D
 50 FOR E = 43 TO 71:
     SET(E,41):
     NEXT E
 60 FOR F = 19 TO 35:
     SET(37,F):
     NEXT F
 70 FOR G = 17 TO 35:
     SET(77,G):
     NEXT G
 80 SET(40,28):
    SET(41,28):
    SET(46,28):
    SET(47,28):
    SET(52,28):
    SET(53,28):
    SET(58,28):
    SET(59,28):
    SET(64,28):
    SET(65,28)
 81 SET(70,28):
    SET(71,28):
    SET(76,28)
 90 FOR H = 40 TO 47:
     SET(35,H):
     NEXT H
100 FOR I = 40 TO 47:
     SET(42,I):
     NEXT I
110 FOR J = 40 TO 47:
     SET(72,J):
     NEXT J
120 FOR K = 40 TO 47:
     SET(79,K):
     NEXT K
130 SET(38,35):
    SET(39,35)
140 FOR L = 37 TO 40:
     SET(L,36):
     NEXT L
150 FOR M = 36 TO 41:
     SET(M,37):
     NEXT M
160 FOR N = 35 TO 42:
     SET(N,38):
     NEXT N
170 SET(75,35):
    SET(76,35)
180 FOR O = 74 TO 77:
     SET(O,36):
     NEXT O
190 FOR P = 73 TO 78:
     SET(P,37):
     NEXT P
```

Program continued

```
200 FOR Q = 72 TO 79:
      SET(Q,38):
      NEXT Q
210 SET(33,26):
    SET(34,26):
    SET(80,26):
    SET(81,26)
220 FOR R = 21 TO 25:
      SET(31,R):
      NEXT R
230 FOR S = 21 TO 25:
      SET(32,R):
      NEXT S
240 FOR T = 21 TO 25:
      SET(82,T):
      NEXT T
250 FOR U = 21 TO 25:
      SET(83,U):
      NEXT U
260 FOR V = 33 TO 36:
      SET(V,21):
      NEXT V
270 FOR W = 78 TO 83:
      SET(W,21):
      NEXT W
280 FOR X = 38 TO 47:
      SET(X,19):
      NEXT X
290 FOR Y = 68 TO 76:
      SET(Y,19):
      NEXT Y
300 FOR Z = 46 TO 52:
      SET(Z,17):
      NEXT Z
310 FOR A = 62 TO 69:
      SET(A,17):
      NEXT A
320 FOR B = 52 TO 55:
      SET(B,14):
      NEXT B
330 FOR C = 60 TO 63:
      SET(C,14):
      NEXT C
340 FOR D = 54 TO 61:
      SET(D,12):
      NEXT D
350 SET(54,13):
    SET(55,13):
    SET(60,13):
    SET(61,13)
360 FOR E = 15 TO 16:
      SET(52,E):
      NEXT E
370 FOR F = 15 TO 16:
      SET(53,F):
      NEXT F
380 FOR G = 15 TO 16:
      SET(62,G):
      NEXT G
390 FOR H = 15 TO 16:
      SET(63,H):
      NEXT H
400 SET(46,18):
    SET(47,18):
    SET(68,18):
    SET(69,18)
410 FOR I = 30 TO 32:
      SET(40,I):
      NEXT I
420 FOR J = 30 TO 32:
      SET(41,J):
```

```
     NEXT J
430 FOR K = 30 TO 32:
     SET(44,K):
     NEXT K
440 FOR L = 30 TO 32:
     SET(45,L):
     NEXT L
450 FOR M = 30 TO 32:
     SET(54,M):
     NEXT M
460 FOR N = 30 TO 32:
     SET(55,N):
     NEXT N
470 FOR O = 30 TO 32:
     SET(60,O):
     NEXT O
480 FOR P = 30 TO 32:
     SET(61,P):
     NEXT P
490 FOR Q = 30 TO 32:
     SET(64,Q):
     NEXT Q
500 FOR R = 30 TO 32:
     SET(65,R):
     NEXT R
510 FOR S = 30 TO 32:
     SET(70,S):
     NEXT S
520 FOR T = 30 TO 32:
     SET(71,T):
     NEXT T
530 FOR U = 30 TO 32:
     SET(74,U):
     NEXT U
540 FOR V = 30 TO 32:
     SET(75,V):
     NEXT V
550 SET(42,30):
    SET(43,30):
    SET(52,30):
    SET(53,30):
    SET(62,30):
    SET(63,30):
    SET(72,30):
    SET(73,30)
560 FOR W = 24 TO 26:
     SET(41,W):
     NEXT W
570 FOR X = 24 TO 26:
     SET(48,X):
     NEXT X
580 FOR Y = 24 TO 26:
     SET(54,Y):
     NEXT Y
590 FOR Z = 24 TO 26:
     SET(61,Z):
     NEXT Z
600 FOR A = 24 TO 26:
     SET(67,A):
     NEXT A
610 FOR B = 24 TO 26:
     SET(74,B):
     NEXT B
620 SET(42,43):
    SET(43,22):
    SET(44,21):
    SET(45,21):
    SET(46,22):
    SET(47,23)
630 SET(55,23):
    SET(56,22):
```

Program continued

```
    SET(57,21):
    SET(58,21):
    SET(59,22):
    SET(60,23)
640 SET(68,23):
    SET(69,22):
    SET(70,21):
    SET(71,21):
    SET(72,22):
    SET(73,23)
650 SET(31,20):
    SET(32,19):
    SET(33,18):
    SET(34,18):
    SET(35,19):
    SET(36,20)
660 SET(78,20):
    SET(79,19):
    SET(80,18):
    SET(81,18):
    SET(82,19):
    SET(83,20)
670 FOR C = 11 TO 16:
     SET(77,C):
     NEXT C
680 FOR D = 11 TO 18:
     SET(38,D):
     NEXT D
690 FOR E = 1 TO 11:
     SET(57,E):
     NEXT E
700 FOR F = 7 TO 11:
     SET(58,F):
     NEXT F
710 FOR G = 58 TO 61:
     SET(G,1):
     NEXT G
720 FOR H = 62 TO 65:
     SET(H,2):
     NEXT H
730 FOR I = 66 TO 69:
     SET(I,3):
     NEXT I
740 FOR J = 70 TO 71:
     SET(J,4):
     NEXT J
750 FOR K = 66 TO 69:
     SET(K,5):
     NEXT K
760 FOR L = 59 TO 64:
     SET(L,6):
     NEXT L
770 FOR M = 32 TO 37:
     SET(M,11):
     NEXT M
780 FOR N = 28 TO 31:
     SET(N,12):
     NEXT N
790 FOR O = 24 TO 27:
     SET(O,13):
     NEXT O
800 FOR P = 28 TO 31:
     SET(P,14):
     NEXT P
810 FOR Q = 32 TO 37:
     SET(Q,15):
     NEXT Q
820 FOR R = 78 TO 87:
     SET(R,11):
     NEXT R
830 FOR S = 86 TO 89:
     SET(S,12):
     NEXT S
840 FOR T = 91 TO 92:
     SET(T,13):
     NEXT T
850 FOR U = 82 TO 89:
     SET(U,14):
     NEXT U
860 FOR V = 78 TO 81:
     SET(V,15):
     NEXT V
870 FOR W = 45 TO 47:
     SET(52,W):
     NEXT W
880 FOR X = 45 TO 47:
     SET(60,X):
     NEXT X
890 SET(53,44):
    SET(54,43):
    SET(55,42):
    SET(56,42):
    SET(57,42):
    SET(58,43):
    SET(59,44)
900 SET(38,41):
    SET(76,41):
    SET(38,44):
    SET(76,44)
910 FOR Y = 42 TO 43:
     SET(37,Y):
     NEXT Y
920 FOR Z = 42 TO 43:
     SET(39,Z):
     NEXT Z
930 FOR A = 42 TO 43:
     SET(75,A):
     NEXT A
940 FOR B = 42 TO 43:
     SET(77,B):
     NEXT B
960 SET(42,23)
970 FOR C = 30 TO 32:
     SET(50,C):
     NEXT C
971 FOR D = 30 TO 32:
     SET(51,D):
     NEXT D
980 SET(32,21):
    SET(53,17)
990 FOR E = 64 TO 68:
     SET(E,43):
     NEXT E
1000 FOR F = 68 TO 70:
     SET(F,44):
     NEXT F
1010 SET(46,44):
     SET(47,44):
     SET(48,45):
     SET(49,45):
     SET(46,46):
     SET(47,46)
1020 GOTO 1020
```

Program Listing 2. *Frankenstein*

```
1 REM   ******** FRANKENSTEIN ********
2 REM   PROGRAMMER: GEORGE JANVIER (SOPHOMORE)
```

Program continued

```
 10 CLS
 20 FOR X = 24 TO 91
 30  SET(X,9):
     SET(X,41)
 40  NEXT X
 50 FOR Y = 9 TO 41
 60  SET(24,Y):
     SET(91,Y)
 65  NEXT Y
 70 FOR Y = 9 TO 11
 80  FOR X = 28 TO 88 STEP 4
 90   SET(X,Y)
100   NEXT X
110  NEXT Y
120 FOR Y = 14 TO 18
130  SET(34,Y):
     SET(45,Y):
     SET(70,Y):
     SET(81,Y)
140  NEXT Y
150 FOR X = 34 TO 45
160  SET(X,14):
     SET(X,18)
170  NEXT X
180 FOR X = 70 TO 81
190  SET(X,14):
     SET(X,18)
200  NEXT X
210 FOR Y = 18 TO 24
220  SET(56,Y):
     SET(60,Y)
230  NEXT Y
240 FOR Y = 24 TO 26
250  SET(53,Y):
     SET(63,Y)
260  NEXT Y
263 FOR X = 53 TO 63
264  SET(X,24):
     SET(X,26)
265  NEXT X
270 FOR Y = 41 TO 47
280  SET(44,Y):
     SET(71,Y)
290  NEXT Y
300 FOR X = 38 TO 77
310  SET(X,31):
     SET(X,37)
320  NEXT X
330 FOR Y = 31 TO 37
340  SET(38,Y):
     SET(77,Y)
350  NEXT Y
360 FOR X = 36 TO 39
370  FOR Y = 15 TO 16
380   SET(X,Y)
390   NEXT Y
400  NEXT X
401 FOR X = 100 TO 125
402  SET(X,23)
403  NEXT X
404 FOR X = 80 TO 125
405  SET(X,33)
406  NEXT X
407 FOR Y = 23 TO 33
408  SET(125,Y)
409  NEXT Y
410 FOR X = 72 TO 75
420  FOR Y = 15 TO 16
430   SET(X,Y)
440   NEXT Y
445  NEXT X
```

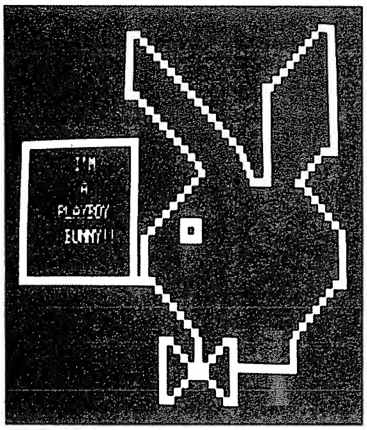
```
446 FOR X = 84 TO 100
447  SET(X,30)
448  NEXT X
449 FOR Y = 23 TO 30
450  SET(100,Y)
451  NEXT Y
452 FOR Y = 25 TO 29
453  SET(104,Y):
     SET(108,Y):
     SET(112,Y):
     SET(116,Y):
     SET(119,Y):
     SET(123,Y)
454  NEXT Y
455 FOR X = 104 TO 108
456  SET(X,25):
     SET(X,29):
     SET(X,27)
457  NEXT X
458 RESET(108,27)
459 FOR X = 112 TO 116
460  SET(X,25):
     SET(X,29)
461  NEXT X
462 FOR X = 119 TO 123
463  SET(X,25):
     SET(X,29)
464  NEXT X
465 SET(80,33):
    SET(81,32):
    SET(82,31):
    SET(83,30)
468 FOR X = 1 TO 100:
     NEXT X
469 FOR X = 38 TO 77
470  RESET(X,31):
     RESET(X,37)
480  NEXT X
490 FOR Y = 31 TO 37
500  RESET(38,Y):
     RESET(77,Y)
510  NEXT Y
520 FOR X = 36 TO 39
530  FOR Y = 15 TO 16
540   RESET(X,Y)
550   NEXT Y
560  NEXT X
570 FOR X = 72 TO 75
580  FOR Y = 15 TO 16
590   RESET(X,Y)
600   NEXT Y
610  NEXT X
620 FOR X = 38 TO 77
630  SET(X,34)
640  NEXT X
650 FOR X = 40 TO 43
660  FOR Y = 16 TO 17
670   SET(X,Y)
680   NEXT Y
690  NEXT X
700 FOR X = 76 TO 79
710  FOR Y = 16 TO 17
720   SET(X,Y)
730   NEXT Y
740  NEXT X
745 FOR X = 1 TO 100:
     NEXT X
750 FOR X = 38 TO 77
760  RESET(X,34)
770  NEXT X
```

Program continued

```
780 FOR X = 40 TO 43
790  FOR Y = 16 TO 17
800   RESET(X,Y)
810   NEXT Y
820  NEXT X
830 FOR X = 76 TO 79
840  FOR Y = 16 TO 17
850   RESET(X,Y)
860   NEXT Y
870  NEXT X
880 GOTO 300
```

Program Listing 3. *Playboy Bunny*

```
  1 REM  ******** PLAYBOY BUNNY ********
  2 REM  PROGRAMMER: DANN SCHWARTZ (SOPHOMORE)
 10 CLS
 20 FOR N = 0 TO 8
 30  SET(28,2 + N)
 40  SET(29,2 + N)
 50  NEXT N
 60 FOR R = 0 TO 9
 70  SET(30 + 2 * R,10 + R)
 80  SET(31 + 2 * R,10 + R)
 90  NEXT R
 91 FOR Y = 0 TO 6
 92  SET(30 + 2 * Y,2 + Y)
 93  SET(31 + 2 * Y,2 + Y)
 94  NEXT Y
100 FOR F = 0 TO 7
110  SET(30 + 2 * N,2 + N)
120  SET(31 + 2 * N,2 + N)
130  NEXT F
140 SET(43,8)
150 FOR G = 0 TO 8
160  SET(44 + 2 * G,9 + G)
170  SET(45 + 2 * G,9 + G)
180  NEXT G
190 FOR T = 0 TO 4
200  SET(60 + T,17 + T)
```



```
I'M
A
PLAYBOY
BUNNY!!!
```

```
210  SET(61 + T,17 + T)
220  NEXT T
230 SET(65,21)
240 SET(66,21)
```

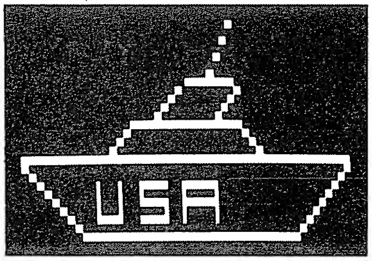
```
245 SET(67,21)
250 FOR X = 0 TO 12
260  SET(68,9 + X)
270  SET(69,9 + X)
280  NEXT X
290 FOR B = 0 TO 8
300  SET(85 - 2 * B,1 + B)
310  SET(86 - 2 * B,1 + B)
320  NEXT B
330 FOR C = 0 TO 15
340  SET(88,2 + C)
350  SET(89,2 + C)
360  NEXT C
361 SET(88,1):
    SET(87,1):
    SET(89,1)
365 SET(87,17)
370 FOR D = 0 TO 4
380  SET(85 - 2 * D,17 + D)
390  SET(86 - 2 * D,17 + D)
400  NEXT D
410 FOR E = 0 TO 6
420  SET(49 - 2 * E,20 + E)
430  SET(50 - 2 * E,20 + E)
440  SET(78 + 2 * E,21 + E)
450  SET(79 + 2 * E,21 + E)
460  NEXT E
465 SET(35,27):
    SET(36,27)
470 FOR F = 0 TO 5
480  SET(33,28 + F)
490  SET(34,28 + F)
500  SET(90,28 + F)
510  SET(91,28 + F)
520  NEXT F
530 FOR H = 0 TO 6
540  SET(35 + 2 * H,33 + H)
550  SET(36 + 2 * H,33 + H)
560  SET(89 - 2 * H,33 + H)
570  SET(88 - 2 * H,33 + H)
580  NEXT H
590 FOR I = 0 TO 2
600  SET(48 ,40 + I)
610  SET(49,40 + I)
620  SET(75,40 + I)
630  SET(76,40 + I)
640  NEXT I
650 FOR J = 0 TO 15
660  SET(60 + J,43)
670  SET(61 + J,43)
680  NEXT J
690 FOR K = 0 TO 7
700  SET(38,40 + K)
710  SET(39,40 + K)
720  SET(58,40 + K)
730  SET(59,40 + K)
740  NEXT K
750 FOR Z = 0 TO 3
760  SET(40 + 2 * Z,40 + Z)
770  SET(41 + 2 * Z,40 + Z)
780  SET(50 + 2 * Z,43 - Z)
790  SET(51 + 2 * Z,43 - Z)
800  SET(50 + 2 * Z,44 + Z)
810  SET(51 + 2 * Z,44 + Z)
820  SET(40 + 2 * Z,47 - Z)
830  SET(41 + 2 * Z,47 - Z)
840  NEXT Z
845 FOR K = 16 TO 32
850  SET(30,K):
     SET(31,K)
```

Program continued

```
855  NEXT K
860 SET(49,44):
    SET(48,43):
    SET(49,43)
870 FOR L = 0 TO 5
880  SET(44 + L,26):
     SET(44 + L,27):
     SET(44 + L,28)
890  NEXT L
891 SET(48,44)
950 PRINT @384, TAB(7);"I'M";
955 PRINT @ 448, TAB(8);"A";
960 PRINT @512, TAB(5);"PLAYBOY";
970 PRINT @ 576, TAB(6);"BUNNY!!";
973 FOR M = 0 TO 32
975  SET(0 + M,33)
977  NEXT M
980 FOR B = 16 TO 32
983  SET(0,B):
     SET(1,B)
985  NEXT B
996 FOR S = 0 TO 30
997  SET(S,16)
999  NEXT S
1100 SET(47,27):
     SET(46,27)
1120 RESET(47,27):
     RESET(46,27)
1130 GOTO 1100
5000 GOTO 5000
```

Program Listing 4. *Ship*

```
 1 REM  ******** SHIP ********
 2 REM  PROGRAMMER: DAVID FUCHS (SOPHOMORE)
 5 CLS
10 FOR X = 34 TO 95:
    SET(X,44):
    NEXT X
20 FOR X = 18 TO 111:
    SET(X ,35):
    NEXT X
30 FOR X = 64 TO 77:
    SET(X,26):
    NEXT X
40 FOR X = 70 TO 71:
    SET(X,25):
    NEXT X
50 FOR X = 72 TO 73:
    SET(X,23):
    NEXT X
```



```
60 FOR X = 74 TO 75:
    SET (X,21):
    NEXT X
```

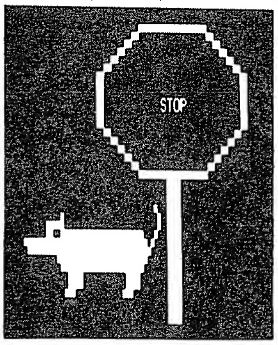
```
 70 FOR X = 76 TO 77:
     SET(X,19):
     NEXT X
130 FOR X = 0 TO 7:
     SET(18 + 2 * X,36 + X):
     SET(19 + 2 * X,36 + X):
     SET(111 - 2 * X,36 + X):
     SET(110 - 2 * X,36 + X):
     NEXT X
140 FOR X = 0 TO 2:
     SET(47 - 2 * X,32 + X):
     SET(46 - 2 * X,32 + X):
     SET(80 + 2 * X,32 + X):
     SET(81 + 2 * X,32 + X):
     NEXT X
150 FOR X = 0 TO 31:
     SET(48 + X,31):
     NEXT X
160 FOR X = 0 TO 3:
     SET(62 - 2 * X,27 + X):
     SET(61 - 2 * X,27 + X):
     SET(78 - 2 * X,27 + X):
     SET(79 - 2 * X,27 + X):
     NEXT X
170 FOR Y = 38 TO 42:
     SET(38,Y):
     NEXT Y
180 FOR Y = 38 TO 42:
     SET(44,Y):
     NEXT Y
190 FOR X = 38 TO 44:
     SET(X,42):
     NEXT X
200 FOR X = 50 TO 57 :
     SET (X,38):
     NEXT X
210 FOR X = 50 TO 57:
     SET(X,40):
     NEXT X
220 FOR X = 50 TO 57:
     SET(X,42):
     NEXT X
230 FOR Y = 38 TO 40:
     SET(50,Y):
     NEXT Y
240 FOR Y = 40 TO 42 :
     SET(57,Y):
     NEXT Y
250 FOR X = 62 TO 71:
     SET (X,38):
     NEXT X
260 FOR X = 62 TO 71:
     SET(X,40):
     NEXT X
270 FOR Y = 38 TO 42 :
     SET(62,Y):
     NEXT Y
280 FOR Y = 38 TO 42:
     SET(71,Y):
     NEXT Y
290 FOR Y = 38 TO 42:
     RESET(38,Y):
     NEXT Y
300 FOR Y = 38 TO 42:
     RESET(44,Y):
     NEXT Y
310 FOR X = 38 TO 44:
     RESET(X,42):
     NEXT X
320 FOR X = 50 TO 57:
     RESET(X ,38):
```

Program continued

```
     NEXT X
 330 FOR X = 50 TO 57:
     RESET(X,40):
     NEXT X
 340 FOR X = 50 TO 57:
     RESET(X,42):
     NEXT X
 350 FOR Y = 38 TO 40:
     RESET(50,Y):
     NEXT Y
 360 FOR Y = 40 TO 42:
     RESET(57,Y):
     NEXT Y
 370 FOR X = 62 TO 71:
     RESET(X,38):
     NEXT X
 380 FOR X = 62 TO 71:
     RESET(X,40):
     NEXT X
 390 FOR Y = 38 TO 42:
     RESET(62,Y):
     NEXT Y
 400 FOR Y = 38 TO 42:
     RESET(71,Y):
     NEXT Y
 410 GOTO 170
5000 GOTO 5000
```

Program Listing 5. *Dog*

```
  1 REM  ******** DOG ********
  2 REM  PROGRAMMER: RANDY KESSLER (SENIOR)
 10 CLS
 20 FOR X = 54 TO 73
 30  Y = 12
 40  SET(X,Y)
 50  NEXT X
 60 SET(52,13)
 70 SET(53,13)
 80 SET(51,14)
 90 SET(50,14)
100 SET(49,15)
```



```
110 SET(48,15)
120 SET(47,16)
130 SET(46,16)
140 SET(45,17)
150 SET(44,17)
160 SET(43,18)
170 SET(42,18)
175 X = 74
```

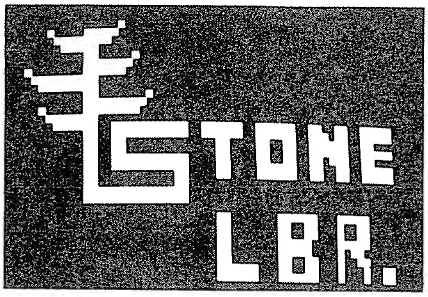
```
180 FOR Y = 13 TO 18
190  FOR D = 1 TO 2
200   SET(X,Y)
210   X = X + 1
220   NEXT D
230  NEXT Y
240 FOR X = 42 TO 43
250  FOR Y = 19 TO 24
260   SET(X,Y)
270   NEXT Y
280  NEXT X
290 FOR X = 84 TO 85
300  FOR Y = 19 TO 24
310   SET(X,Y)
320   NEXT Y
330  NEXT X
340 X = 44
350 FOR Y = 25 TO 30
360  FOR D = 1 TO 2
370   SET(X,Y)
375   X = X + 1
380   NEXT D
390  NEXT Y
400 FOR X = 54 TO 73
410  Y = 30
420  SET(X,Y)
430  NEXT X
435 X = 83
440 FOR Y = 25 TO 29
450  FOR D = 1 TO 2
460   SET(X,Y)
470   X = X - 1
480   NEXT D
490  NEXT Y
500 FOR Y = 31 TO 47
510  FOR X = 62 TO 65
520   SET(X,Y)
530   NEXT X
540  NEXT Y
560 PRINT @478,"STOP";
600 SET(58,34)
610 SET(59,35)
620 SET(59,36)
630 SET(58,37)
640 SET(57,38)
650 SET(56,39)
660 FOR Y = 38 TO 40
670  X = 55
680  SET(X,Y)
690  NEXT Y
700 FOR Y = 37 TO 41
710  FOR X = 32 TO 54
720   SET(X,Y)
730   NEXT X
740  NEXT Y
750 SET(32,35)
760 SET(32,36)
770 FOR X = 28 TO 32
780  Y = 36
790  SET(X,Y)
800  NEXT X
810 SET(28,37)
820 SET(29,37)
830 FOR X = 23 TO 31
840  FOR Y = 38 TO 39
850   SET(X,Y)
860   NEXT Y
870  NEXT X
880 SET(22,38)
890 SET(31,40)
900 FOR X = 34 TO 37
```

Program continued

```
 910   FOR Y = 42 TO 43
 920    SET(X,Y)
 930    NEXT Y
 940   NEXT X
 950  FOR X = 33 TO 35
 960   Y = 44
 970   SET(X,Y)
 980   NEXT X
 990  FOR X = 50 TO 53
1000   FOR Y = 42 TO 43
1010    SET(X,Y)
1020    NEXT Y
1030   NEXT X
1040  SET(49,44)
1050  SET(50,44)
1060  SET(51,44)
1070  FOR X = 1 TO 500:
       NEXT X
1071  FOR K = 62 TO 57 STEP - 1
1073   SET(K,44)
1075   J = 1
1080   SET(56,40)
1090   FOR X = 1 TO 60:
        NEXT X
1100   SET(57,41)
1110   FOR X = 1 TO 100:
        NEXT X
1120   RESET(56,40)
1130   SET(58,42)
1140   FOR X = 1 TO 100:
        NEXT X
1150   RESET(57,41)
1160   SET(59,43)
1170   FOR X = 1 TO 100:
        NEXT X
1200   RESET(59,43)
1210   RESET(58,42)
1212   SET(K,44)
1215   PRINT @950,"PLOP!";
1217   FOR X = 1 TO 200:
        NEXT X
1218   PRINT @950,"       ";
1225   NEXT K
1230  SET(59,44)
1240  PRINT @945,"WHAT A MESS!";
1250  PRINT @478,"PHEW ";
2222  GOTO 2222
```

Program Listing 6. *Stone Lumber logo*

```
1 REM   ******** STONE LUMBER ********
2 REM   PROGRAMMER: BOB STONE (SENIOR)
```



```
10 CLS
20 SET (42,4):
   SET (43,4)
30 FOR Y = 5 TO 26
40  SET (32,Y):
    SET(33,Y):
    SET(34,Y):
    SET (35,Y)
50  NEXT Y
60 FOR Y = 17 TO 22
70  SET (40,Y):
    SET(41,Y):
    SET (42,Y):
    SET (43,Y)
80  NEXT Y
90 FOR Y = 21 TO 26
100  SET (58,Y):
     SET (59,Y):
     SET (60,Y):
     SET (61,Y)
110  NEXT Y
120 FOR Y = 17 TO 23
130  SET (70,Y):
     SET (71,Y):
     SET (72,Y):
     SET (73,Y):
     SET(82,Y):
     SET (83,Y):
     SET (84,Y):
     SET (85,Y)
140  SET (90,Y):
     SET (91,Y):
     SET (92,Y):
     SET (93,Y):
     SET (98,Y):
     SET (99,Y):
     SET (100,Y):
     SET (101,Y)
145  SET (106,Y):
     SET (107,Y):
     SET (108,Y):
     SET (109,Y):
     SET(114,Y):
     SET (115,Y):
     SET (116,Y):
     SET (117,Y)
150  NEXT Y
160 FOR X = 32 TO 41
170  SET (X,5)
180  NEXT X
190 FOR X = 24 TO 39
200  SET(X,6)
210  NEXT X
220 SET (22,5):
    SET (23,5)
230 FOR X = 26 TO 31
240  SET (X,7)
250  NEXT X
260 FOR X = 36 TO 45
270  SET (X,9)
280  NEXT X
290 FOR X = 36 TO 43
300  SET (X,10)
310  NEXT X
320 FOR X = 18 TO 31
330  SET (X,11)
340  NEXT X
350 FOR X = 20 TO 31
360  SET (X,12)
370  NEXT X
380 FOR X = 36 TO 49
```

Program continued

```
390  SET (X,14)
400  NEXT X
410 FOR X = 36 TO 47
420  SET (X,15)
430  NEXT X
440 SET (16,10):
    SET (17,10)
450 FOR X = 22 TO 31
460  SET (X,16)
470  NEXT X
480 FOR X = 24 TO 31
490  SET (X,17)
500  NEXT X
510 SET (46,8):
    SET (47,8)
520 SET (50,13):
    SET (51,13)
530 SET (20,15):
    SET (21,15)
540 FOR X = 44 TO 61
550  SET (X,17):
     SET (X,18)
560  NEXT X
570 FOR X = 44 TO 57
580  SET (X,21):
     SET (X,22)
590  NEXT X
600 FOR X = 36 TO 57
610  SET (X,25):
     SET (X,26)
620  NEXT X
630 FOR X = 66 TO 77
640  SET (X,17):
     SET (X,18)
650  NEXT X
660 FOR X = 86 TO 89
670  SET (X,17):
     SET (X,18):
     SET (X,22):
     SET (X,23)
680  NEXT X
690 FOR X = 102 TO 103
700  SET (X,18):
     SET (X,19):
     SET (X,20)
710  NEXT X
720 FOR X = 104 TO 105
730  SET (X,19):
     SET (X,20):
     SET (X,21)
740  NEXT X
750 FOR X = 118 TO 123
760  SET (X,17):
     SET (X,18)
770  SET (X,20)
780  SET (X,22):
     SET (X,23)
790  NEXT X
800 FOR Y = 28 TO 35
810  SET (70,Y):
     SET (71,Y):
     SET (72,Y):
     SET (73,Y)
820  SET (88,Y):
     SET (89,Y):
     SET (90,Y):
     SET (91,Y)
825  SET (96,Y):
     SET(97,Y)
830  SET (106,Y):
     SET (107,Y)
```

```
840  NEXT Y
850 FOR X = 74 TO 81
860  SET (X,34):
     SET (X,35)
870  NEXT X
880 FOR X = 92 TO 99
890  SET (X,28):
     SET (X,35)
900  NEXT X
910 FOR Y = 31 TO 32
920  SET (92,Y):
     SET (93,Y):
     SET (94,Y):
     SET (95,Y)
930  NEXT Y
940 FOR Y = 29 TO 30
950  SET (98,Y):
     SET (99,Y)
960  NEXT Y
970 FOR Y = 33 TO 34
980  SET(98,Y):
     SET (99,Y)
990  NEXT Y
1000 SET (108,28):
     SET (109,28):
     SET (110,28):
     SET (111,28)
1010 SET (108,31):
     SET (109,31)
1020 FOR Y = 28 TO 33
1030  SET (112,Y):
      SET (113,Y)
1040  NEXT Y
1050 FOR Y = 28 TO 31
1060  SET (114,Y):
      SET (115,Y)
1070  NEXT Y
1080 FOR Y = 31 TO 32
1090  SET (110,Y):
      SET (111,Y)
1100  NEXT Y
1110 FOR Y = 33 TO 35
1120  SET (114,Y):
      SET (115,Y)
1130  NEXT Y
1140 FOR Y = 34 TO 35
1150  SET (120,Y):
      SET (121,Y):
      SET (122,Y):
      SET (123,Y)
1160  NEXT Y
1200 GOTO 1200
```

GAMES

Asteroid Adventure
Compukala
Puzzler

Asteroid Adventure

by Greg Perry and Don Taylor

Despite widely held beliefs to the contrary, it is possible to write enjoyable programs for the TRS-80 Level II with only 4K of memory. Asteroid Adventure is a prime example. The object of this space game is to guide your ship through the asteroids of space, land on the moon safely, and have fuel to spare.

At the beginning of the program, you are asked to enter your experience level on a scale of one to ten. One is for advanced players, and ten is for beginners. The first time you play you might enter a five. After you've selected a level, the screen clears and a field of asteroids (*s) is printed. On the right-hand side of the screen a half-moon appears; in the lower right corner a fuel reading, based on your experience level, is printed.

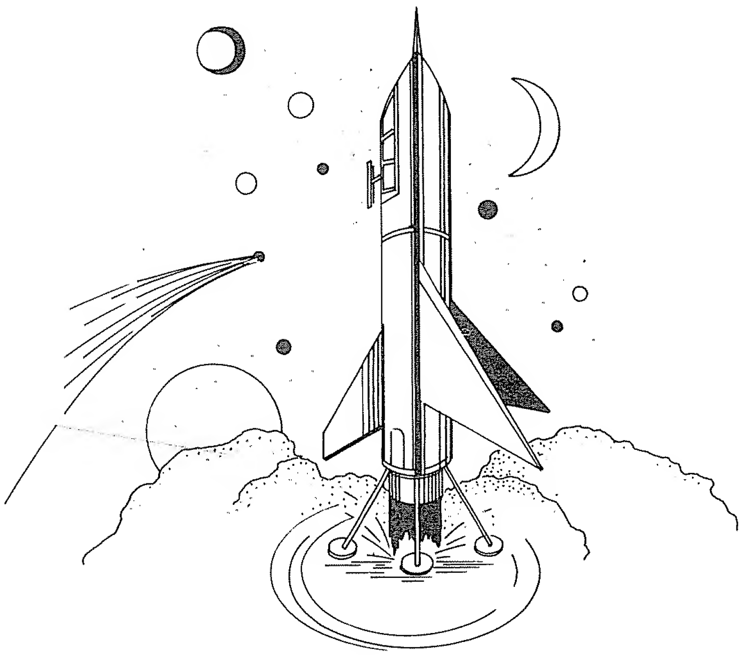
Your space ship is the greater-than sign in the upper left corner. Use the four arrow keys to guide your ship. Holding a key down causes continuous movement. The game commences when the first direction key is pressed. Fuel consumption begins at the moment you first press a key. With skill and luck you can maneuver your ship to the moon and land safely. If you hit an asteroid during your voyage, you will blow up. Remember to keep an eye on your fuel reading; you wouldn't want to be caught in space with no gas!

Lines:	Explanation:
30–115	Instructions
190–220	Input directions for movement
240–270	Land, blowup, or move
370–410	Draw random asteroid field
420–520	Draw moon
530–540	Blowup
600–610	Ran out of fuel
620–680	Beginning and experience input

Table 1. *Program blocks*

About the Program

The asteroid field is printed using the random-number generator; so the field is different every play. Your ship is moved by using the PEEK statements starting at line 190. Hitting an asteroid or landing on the moon is detected by a PEEK that scans the next screen location in the direction of the current movement. The speed of the game, as well as the amount of fuel you have, is determined by the experience level you choose.



Program Listing. *Asteroid Adventure*

Encyclopedia Loader

```
 1Ø CLS
 2Ø GOSUB 62Ø
 3Ø PRINT "iNSTRUCTIONS?":
    P$ = ""
 4Ø P$ = INKEY$:
    IF P$ = "" GOTO 4Ø
 5Ø IF P$ = "N" GOTO 12Ø
 6Ø IF P$ < > "Y" GOTO 4Ø
 7Ø PRINT " YOU ARE THE CAPTAIN OF A STAR SHIP.  YOU HAVE TO "
 8Ø PRINT " SAFELY GUIDE YOUR CREW THROUGH THE ASTEROIDS TO "
 9Ø PRINT " THE MOON'S SURFACE BEFORE THE FUEL RUNS OUT.  USE THE "
1ØØ PRINT " ARROW KEYS TO GUIDE YOUR SHIP.  HOLDING THE KEYS "
11Ø PRINT " DOWN WILL CAUSE CONTINUOUS MOVEMENT.  PRESS ENTER TO "
111 PRINT " START "
115 A$ = INKEY$:
    IF A$ = ""
     THEN
      GOTO 115 :
     ELSE
      GOTO 12Ø
12Ø P = Ø:
    T = (E - 1) * 1Ø + 9Ø:
    N = Ø
13Ø CLS
14Ø GOSUB 39Ø
15Ø GOSUB 43Ø
16Ø PRINT @P,">";
17Ø PRINT @96Ø,"FUEL:";T;
18Ø REM * INPUT MOVE
19Ø IF PEEK(144ØØ) = 8
     THEN
      AD = - 64:
      N = 1
2ØØ IF PEEK(144ØØ) = 16
     THEN
      AD = 64:
      N = 1
21Ø IF PEEK(144ØØ) = 64
     THEN
      AD = 1:
      N = 1
22Ø IF PEEK(144ØØ) = 32
     THEN
      AD = - 1:
      FG = 1:
      N = 1
23Ø IF N = Ø AND AD = Ø
     THEN
      19Ø
24Ø IF P + AD < Ø OR P + AD > 1Ø23
     THEN
      AD = Ø
25Ø IF PEEK(1536Ø + AD + P) = 42
     THEN
      GOTO 53Ø:
      REM  BLOW-UP
26Ø IF FG = 1
     THEN
      IF PEEK(1536Ø + AD + P) > = 129
       THEN
        AD = Ø:
        GOTO 28Ø
27Ø IF PEEK(1536Ø + AD + P) > = 129
     THEN
      GOTO 55Ø:
      REM  MOON
28Ø PRINT @P,".";:
    Q = Q + 1
```

Program continued

```
290 PRINT @P + AD,">";
300 P = P + AD
310 T = T - 1:
    IF T < = 0
     THEN
      GOTO 600:
     ELSE
      PRINT @965,T;
320 AD = 0
330 FG = 0
340 REM * SPEED FACTOR
350 FOR X = 1 TO E * 10:
     NEXT X
360 GOTO 190
370 REM * SET STAR FIELD
380 RANDOM
390 CR = 0:
400 R = RND(10) + 2:
    CR = CR + R:
    IF CR > 1022
     THEN
      RETURN
410 PRINT @ CR,"*";:
    GOTO 400
420 REM * DRAW MOON
430 X = 63
440 FOR I = 1 TO 8
450  PRINT @X, STRING$(I, CHR$(191));
460  X = X + 63:
     NEXT I
470 X = X + 1
480 FOR I = 8 TO 2 STEP - 1
490  PRINT @X, STRING$(I, CHR$(191));
500  X = X + 65:
     NEXT I
510 FOR Y = 45 TO 47:
     FOR X = 126 TO 127:
      SET(X,Y):
      NEXT X,Y
520  RETURN
530  CLS :
     PRINT CHR$ (23)
540  FOR I = 1 TO 150:
      PRINT @ RND(1000),"* B O O M !!! *";:
      NEXT I:
     GOTO 690
550  CLS :
     PRINT CHR$(23):
     FOR I = 1 TO 10 * PRINT @272,"MISSION SUCCESFUL":
      FOR P1 = 1 TO 50:
       NEXT P1
560   PRINT @272, CHR$(30):
      FOR P1 = 1 TO 50:
       NEXT P1:
      NEXT I
570  GOTO 690
600  CLS :
     PRINT :
     PRINT :
     PRINT "SORRY BUT YOU JUST RAN OUT OF FUEL (SPACE IS TOUGH ISN'T
      IT)":
     FOR P6 = 1 TO 900:
      NEXT P6:
     GOTO 690
610  GOTO 690
620  CLS
630  PRINT CHR$(23):
     PRINT :
     PRINT :
     PRINT :
     PRINT "     ASTEROID ADVENTURE"
```

```
640  FOR I = 1 TO 200:
      NEXT I
650  CLS :
     PRINT :
     PRINT :
     PRINT :
     PRINT :
     PRINT "WHAT IS YOUR EXPERIENCE LEVEL"
660  INPUT "<1-ADVANCED TO 10 BEGINNER >";E
670  IF E < 1 OR E > 10
      THEN
       CLS :
       PRINT :
       PRINT :
       GOTO 660
680  RETURN
690  CLS :
     PRINT :
     PRINT "DO YOU WANT TO PLAY AGAIN?"
700  A$ = ""
710  A$ = INKEY$:
     IF A$ = ""
      THEN
       710
720  IF A$ = "Y"
      THEN
       RUN
730  IF A$ < > "N"
      THEN
       710
740  CLS :
     PRINT CHR$(23):
     PRINT @210,"THANK YOU"
750  PRINT @336,"FOR PLAYING"
760  FOR L = 1 TO 900:
      NEXT L
770  CLS :
     FOR I = 1 TO 7:
      PRINT CHR$(23):
      PRINT @268,"ASTEROID ADVENTURE":
      FOR P1 = 1 TO 100:
       NEXT P1
775   IF I = 7 GOTO 790
780   PRINT @268, CHR$(30):
      FOR T = 1 TO 70:
       NEXT T:
      NEXT I
790  FOR I = 1 TO 1000:
      NEXT I:
     CLS
800  CLS
```

Compukala

by Peter K. Moller

Today the game of Kala is played on an oblong or circular surface about the size of a breadboard, but the game has a long history. Even before the Phoenicians were reading and writing, Kala was being played by burly competitors who rolled boulders around huge, open fields.

Compukala, which is probably the latest update on this game, is played within the less dangerous limits of the TRS-80. The opposition can be either another player or the not-so-burly TRS-80 itself.

Kala: Pips in Pits

To understand how the game is played, you must first be familiar with the geography of the board. (See Figure 1.) On each side of the board are six indentations called pits, numbered 1 through 6 on each side in opposite directions. At each end of the board are larger indentations—one for each player—which are called the kala. When you play on a board, the pips, or playing pieces, can be small, simple objects such as matches, coins, or dried beans. At the beginning of a round, each player has the same number of pips in each of his six pits. The two kala are empty. The players determine between themselves how many pips will be in each pit at the start. The complexity of the game depends on the number of pips used.

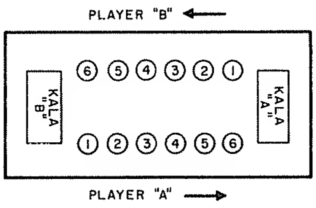


Figure 1. *The Kala board. Moves are made in a counterclockwise direction.*

Players move by picking up all the pips in a pit and placing one in each successive pit, including the kala, in a counterclockwise rotation, until all the pips have been redeposited. You'll often find yourself placing your pips into your opponent's pits. This is part of the strategy of Kala.

The object of the game is for one player to accumulate as many pips as possible on his side of the board *and* in his kala. A round is over when one of the players is left without any pips on his side of the board and, therefore, cannot move. Scores are determined by the total number of pips each player retains at the end of a round. The end of the match occurs when one player accumulates a mutually agreed upon total of pips.

The rules of Kala are surprisingly simple for a game which requires the strategy of backgammon and returns the endless permutations of cribbage. There are only three rules:

- A repeat turn is earned when the last pip of any player's move lands in *his* kala.
- If a player's last pip lands in an empty pit on his side of the board, any pips in the pit directly opposite are captured and deposited in the captor's kala along with the capturing pip.
- Once a pip is placed in a kala, it remains there until the end of the round.

Compukala: No Pips, No Pits

When you load and run this computer version of Kala, the computer prompts you to enter the number of pips that will determine the end of a match. This can be any number, depending on how many rounds of the game you wish to play. Next, enter the number of pips to start the round. In this version, the number of pips available ranges from two to five to allow for beginning and advanced levels of play. The program then asks for the players' names. If you wish to play against the computer, enter *computer* as player number 2.

A random number determines which player moves first. From that point on in the round, plays are made in turn. Each player enters his move by inputting the pit number from which he wishes to move. After each turn, the computer displays the distribution of the board and updates the scores. It also traps illegal moves (such as moving from an empty pit or entering a number larger than 6) and prompts the player to make another entry. If at any time you wish to refresh your memory about the pit number designations, enter the letter P, and the board numbers will be displayed. Enter a P again to return the display to the pip distribution mode.

Compukala is written in Disk BASIC and requires less than 16K of RAM. You can delete error traps and features which provide a clear and symmetrical display if memory overhead is critical. The residue function defined in line 80 gives the game its circular nature. Once the fourteenth position of the vector, A(14), has been reached, the first position of the vector is returned.

If you decide to play against the computer, you will find it has a low level of intelligence. The computer knows only the rules of the game, and its only strategy is to attempt to empty pits 4, 5, and 6 in its first two moves. Once you discover the subtleties of the game, you can make the computer a more worthy opponent by programming it with more strategies.

The word kala is taken from Jainist philosophy, a reformist sect of Hinduism founded in the sixth century B.C. It signifies time, the eternal aspect of existence. There couldn't be a better name for this game; you'll find its pleasures are endless.

Program Listing. *Compukala*

Encyclopedia Loader

```
 10 :
    '
 20 :
    '                    COMPUKALA BY PETER K. MOLLER
 25 :
    '                              7/1/80
 30 :
    '
 40 :
    '                     **INITIALIZE
 50 :
    '
 60 CLS :
    CLEAR 100
 70 DIM A(14)
 80 DEF FN RES(A,B) = A - ( INT(A / B) * B)
 90 GOSUB 740
100 PRINT @130,"ENTER MAXIMUM NUMBER OF PIECES FOR END OF MATCH-----
    >";:
    INPUT PS
110 PRINT @130,"ENTER NUMBER OF PIECES (BETWEEN 2 AND 5) TO BEGIN---
    -->";:
    INPUT P
120 IF P > 5 OR P < 2 PRINT @70, CHR$(30):
    GOTO 110
130 GOSUB 740 :
    GOSUB 1970
140 IF G = > 1 GOTO 240
150 :
    '
160 :
    '                  **INPUT PLAYER NAMES
170 :
    '
180 PRINT @128, CHR$(30)
190 PRINT @141,"ENTER NAME OF PLAYER #1==>";:
    LINE INPUT N$(0)
200 PRINT @141, CHR$(30)
210 PRINT @141,"ENTER NAME OF PLAYER #2==>";:
    LINE INPUT N$(1)
220 PRINT @141, CHR$(30)
230 :
    '
240 :
    '                  **DETERMINE FIRST MOVE
250 :
    '
260 I = RND(2) - 1
270 GOTO 350
280 :
    '
290 :
    '                  **EXCHANGE PLAYERS
300 :
    '
310 IF I = 1
      THEN
        I = 0 :
      ELSE
        I = 1
320 :
    '
330 :
    '                  **GOSUB BOARD ROUTINE AND SCORES
340 :
    '
350 GOSUB 790 :
```

Program continued

```
    GOSUB 920
360 :
370 :
                         **INPUT AND COMPUTE MOVES
380 :
390 IF N$(1) = "COMPUTER" AND I = 1 GOSUB 1710
400 PRINT @75,"     ENTER NUMBER OF PIT TO EMPTY.       ";
410 M$ = INKEY$:
    IF M$ = "" GOTO 410
420 PRINT @64, CHR$(30);:
    PRINT @192, CHR$(30);
430 IF M$ = "P" GOSUB 1000 :
    GOTO 350
440 PRINT @78,"     ";N$(I);" MOVES FROM PIT # ";M$;
450 M = VAL(M$)
460 IF M > 6 OR M < 1
     THEN
      GOSUB 1180
470 IF I = 1
     THEN
      M = M + 7
480 IF A(M) = 0
     THEN
      GOSUB 1180
490 IF I = 0 GOSUB 1860 :
     ELSE
      GOSUB 1910
500 D = FN RES(M + A(M),14)
510 A(M) = 0
520 C = M
530 C = C + 1
540 IF C > 13
     THEN
      C = 0
550 A(C) = A(C) + 1
560 IF C < > D
     THEN
      530
570 GOSUB 790
580 :
590 :
                         **CHECK FOR ENDGAME
600 :
610 IF I = 0 AND C > = 7 OR I = 1 AND C < = 7 OR C = 0 OR C
    = 7 GOTO 630
620 IF A(C) = 1 AND A(14 - C) = > 1 GOSUB 1240
630 IF A(1) + A(2) + A(3) + A(4) + A(5) + A(6) = 0 GOSUB 1380
640 IF A(8) + A(9) + A(10) + A(11) + A(12) + A(13) = 0 GOSUB 1380
650 :
660 :
                         **CHECK FOR REPEAT TURN
670 :
680 IF (C = 0 AND I = 1) OR (C = 7 AND I = 0) GOSUB 1650
690 :
700 :
                         **NEXT TURN
710 :
720 GOTO 310
730 :
740 :
                         **PAINT BORDER
```

```
 750 :
     '
 760 PRINT @0, STRING$(63,191);:
     PRINT @256, STRING$(63,191);:
     PRINT @384, STRING$(63,179);
 770 RETURN
 780 :
     '
 790 :
     '                      **DRAW THE BOARD
 800 :
     '
 810 PRINT @205,"  =ENTER 'P' TO SEE PIT NUMBERS=  ";
 820 PRINT @710,A(0);:
     PRINT @754,A(7);
 830 PRINT @847,A(1);:
     PRINT @852,A(2);:
     PRINT @857,A(3);:
     PRINT @862,A(4);:
     PRINT @867,A(5);:
     PRINT @872,A(6);
 840 PRINT @616,A(8);:
     PRINT @611,A(9);:
     PRINT @606,A(10);:
     PRINT @601,A(11);:
     PRINT @596,A(12);:
     PRINT @591,A(13);
 850 IF I = 0 PRINT @921,N$(0);"--->"; :
      ELSE
       GOTO 880
 860 PRINT @535, CHR$(30);
 870 GOTO 900
 880 PRINT @535,"<---";N$(1);
 890 PRINT @921, CHR$(30);
 900 RETURN
 910 :
     '
 920 :
     '                      **KEEP SCORES
 930 :
     '
 940 T1 = 0:
     T2 = 0
 950 T1 = A(1) + A(2) + A(3) + A(4) + A(5) + A(6) + A(7)
 960 T2 = A(8) + A(9) + A(10) + A(11) + A(12) + A(13) + A(0)
 970 PRINT @330,N$(0);"'S SCORE=";T1;:
     PRINT @353,N$(1);"'S SCORE=";T2;
 980 RETURN
 990 :
     '
1000 :
     '                      **DISPLAY BOARD NUMBERS
1010 :
     '
1020 PRINT @711,"KALA";:
     PRINT @753,"KALA";
1030 FOR W = 847 TO 872 STEP 5
1040  Q = Q + 1
1050  PRINT @W,Q;
1060  NEXT W
1070 U = 7
1080 FOR W = 591 TO 616 STEP 5
1090  U = U - 1
1100  PRINT @W,U;
1110  NEXT W
1120 Q = 0
1130 PRINT @73,"   TO RETURN TO GAME BOARD....ENTER 'P'.";
1140 E$ = INKEY$:
     IF E$ = "" GOTO 1140
1150 PRINT @704, CHR$(30)
```

Program continued

```
1160 GOTO 790
1170 :
     '
1180 :
     '                    **ILLEGAL MOVE ROUTINE
1190 :
     '
1200 PRINT @78,"ILLEGAL MOVE.  PLEASE ENTER AGAIN!!!";
1210 FOR TT = 1 TO 550:
      NEXT
1220 GOTO 350
1230 :
     '
1240 :
     '                    **CAPTURE ROUTINE
1250 :
     '
1260 FOR TT = 1 TO 50
1270  PRINT @728,"**CAPTURE**";
1280  PRINT @728,"            ";
1290  NEXT TT
1300 IF I = 1 GOTO 1340
1310 A(7) = A(C) + A(14 - C) + A(7)
1320 A(C) = 0:
     A(14 - C) = 0
1330 GOTO 1360
1340 A(0) = A(C) + A(14 - C) + A(0)
1350 A(C) = 0:
     A(14 - C) = 0
1360 GOTO 790
1370 :
     '
1380 :
     '                    **END OF ROUND ROUTINE
1390 :
     '
1400 G = G + 1
1410 FOR TT = 1 TO 50
1420  PRINT @711,"   <<  GAME IS OVER!  HIGH SCORE WINS!  >>";
1430  PRINT @711, CHR$(30);
1440  NEXT TT
1450 GOSUB 920
1460 PRINT @80,"           ROUND";G;"                  ";
1470 IF T1 > T2
      THEN
       N4$ = N$(0) :
      ELSE
       N4$ = N$(1)
1480 PRINT @330,N$(0);" ENDS WITH:";T1;:
     PRINT @353,N$(1);" ENDS WITH:";T2;
1490 TS = T1 + TS:
     TQ = T2 + TQ
1500 PRINT @133,N$(0);" PIECES WON=";TS;:
     PRINT @165,N$(1);" PIECES WON=";TQ;
1510 FOR TT = 1 TO 100
1520  PRINT @206,N4$;" WON THIS ROUND OF THE GAME!!";
1530  PRINT @206, CHR$(30);
1540  NEXT TT
1550 IF TS = > PS OR TQ = > PS GOTO 1610
1560 PRINT @717,"TO PLAY ANOTHER ROUND, HIT 'ENTER'.";:
     LINE INPUT E$
1570 K = 0:
     CLS :
     GOTO 130
1580 :
     '
1590 :
     '                    **END OF MATCH ROUTINE
1600 :
     '
1610 IF TS > TQ
```

```
      THEN
       N4$ = N$(0) :
      ELSE
       N4$ = N$(1)
1620 PRINT @711,"     ";N4$;" HAS WON THE KALA MATCH!  CONGRATS!
            ";
1630 PRINT @970,"FOR ANOTHER MATCH, SIMPLY HIT ENTER.";:
     LINE INPUT Z$
1635 RUN
1640 :
     '
1650 :
     '                  **REPEAT TURN ROUTINE
1660 :
     '
1670 IF N$(1) = "COMPUTER" AND I = 1 GOTO 1750
1680 PRINT @78,"     ";N$(I);", TAKE ANOTHER TURN!!   ";
1690 GOTO 410
1700 :
     '
1710 :
     '                   **THE COMPUTER'S MOVES
1720 :
     '
1730 PRINT @205, CHR$(30);
1740 K = K + 1
1750 M = RND(14)
1760 IF K < 3 AND M < 11 GOTO 1750
1770 IF M = > 1 AND M < = 7 GOTO 1750
1780 IF A(M) = 0 GOTO 1750
1790 PRINT @76,"     ";N$(1);" MOVES FROM PIT #";M - 7;
1800 GOSUB 1910
1810 FOR TT = 1 TO 50:
      NEXT TT
1820 GOTO 500
1830 :
     '
1840 :
     '                  **MOVE INDICATORS ROUTINE
1850 :
     '
1860 FOR TT = 1 TO 50
1870  PRINT @844 + (M * 5),">";
1880  PRINT @844 + (M * 5)," ";
1890  NEXT TT
1900 RETURN
1910 FOR TT = 1 TO 50
1920  PRINT @621 - (M * 5) + 35,"<";
1930  PRINT @621 - (M * 5) + 35," ";
1940  NEXT TT
1950 RETURN
1960 :
     '
1970 :
     '                  **INITIALIZE POSITIONS ROUTINE
1980 :
     '
1990 FOR X = 0 TO 13:
      A(X) = P:
      NEXT
2000 A(0) = 0:
     A(7) = 0
2010 RETURN
```

Puzzler

by James P. Morgan

Word-finder puzzles appear daily in newspapers and monthly in a variety of puzzle magazines. The following program presents one method of creating these popular pastimes with your TRS-80 or any other micro with a compatible language structure. The program is written in Level II and requires a 16K memory.

Program Operation

After you load from tape and enter RUN, a brief explanation of the game appears on the video. While you read this information, the computer forms a 12 × 19 array of random letters. Then, specific words selected from a data bank are inserted into the array horizontally, vertically, and diagonally in random order. This process takes 20 to 25 seconds. Press any key to clear the screen and display the puzzle with the eight hidden words. A list of the eight words will appear below the puzzle. To play, search through the jumble of letters for one of the hidden words. Once you have located one, type the word and press ENTER. The program then prompts you for the coordinates, vertical and horizontal, of each letter in the word. As you enter each letter, the computer looks for a corresponding match-up on the screen. If it finds one, it redraws the screen with that letter missing. If the match is incorrect, the game ends, and the program returns to the beginning.

The first bit of string manipulation begins at line 320. The LEN(string) function extracts the number of letters in a given string and returns a numerical value. This value is assigned to the variable N and is used for subsequent string manipulations. Lines 330 and 340 determine randomly the manner in which the words are inserted in the puzzle. For example, if the value of L returns as 1, line 340 sends the program to subroutine 400, the first option in the line. Assuming the program jumps to line 400, let's follow the logic of just how the program inserts a word into the puzzle from that point.

Subroutines 400 through 1100 all work in the same manner. Line 410 chooses a starting location at random. Lines 420 through 450 then test to find space to insert the selected word. Line 420 sets up a counter which corresponds to the length of the selected word. The next two lines test to find if: (1) the proposed location is outside the confines of the array; and (2) the proposed location is already filled with another letter. If the answer to either question is yes, the program goes back to line 410 for another starting point. If all of the proposed spaces are within the array and are unoccupied, the program moves on to lines 460 through 490 to insert the word. The I counter establishes the number of letters to be inserted. The value of the location A$(XI,Y) is assigned to each letter in turn. Continue to play until you discover all eight words and remove them from the array. You can then opt to continue with a new puzzle or end the game.

How the Program Works

Although using Puzzler is quite simple, the programming techniques used to create the puzzle are a bit more complex. Lines 10 through 110 print general information concerning the puzzle. As you read this information, the program moves on to the computations required to build the structure of the puzzle. The three arrays used in the puzzle are DIMensioned at line 200. The A$(X,Y) array used throughout forms the visual display. The C$(Z) array stores 50 words, of which eight are selected at random to insert into the A$(X,Y) array. The W$(A1) array stores the eight words selected from the C$(Z) array. The POKE statement in this line is required for proper execution of the READ function of the TRS-80, Revision G.

Lines 210 through 250 initially set the 12 × 19 array to a value of empty spaces two columns wide. These empty spaces are eventually filled either by random letters or by the specific letters of the puzzle words. Lines 260 through 280 fill the C$(Z) array with the words from the data bank. Lines 290 through 350 begin the process of selecting the words and inserting them into the A$(X,Y) array. A1 is initially set at 0 and serves as a counter in later subroutines to track the eight selected words. Line 300 selects a random number which is then used to transfer the word value of that number into the W$(A1) array at line 310.

The letter to be placed in the puzzle is derived from the MID$ function which works as follows: The word is identified by the W$(A1) array which was determined at line 310. The next two values in the MID$ statement find the specific letter. I + 1 identifies the starting point in the W$(A1) string, while the last value, 1, indicates that only one letter is to be extracted. As the counter increments, the letters are pulled one by one from the selected word and are placed in order in the puzzle array. Line 490 keeps track of the number of words inserted, and when the total reaches eight, the program branches to line 1300. Otherwise, it returns to line 300, selects another word, and repeats the placement process.

Two Loops

Lines 1300 to 1390 complete the job of filling in the entire A$(X,Y) array. Two loops accomplish this task. As the program increments through the Y,X loops, the test at line 1330 checks each array location for occupancy. If there is already a letter assigned to a specific location, the program jumps to the next value of Y,X. If the location is empty, lines 1340 through 1360 fill it in with a random letter.

Since the ASCII codes for the letters of the alphabet run from 65 through 90, a random number from 1 through 26, when added to 64, results in a number corresponding to the alphabetical portion of the code. Line 1360 translates that number via CHR$(n), which returns the letter corresponding to the number and assigns it to the A$(X,Y) location. The array is now complete. It takes about 20 to 25 seconds for the computer to create the filled-in puzzle. At this point, you can see the results by hitting any key. The short routine which accomplishes this task is found in line 1390. For those who have not discovered this powerful tool, a word of explanation is in order. Line 1390:

```
M$ = INKEY:IF M$ = " " THEN 1390
```

works as follows: INKEY$ tells the computer to stand by, because the user is going to enter something directly from the keyboard. If " " (nothing) is input, the statement tells the program to go back to the beginning of line 1390 and wait for the person at the keyboard to do something. When you press any key, INKEY$ assigns the key value to the string variable M$, and the program moves on to the next line—simple yet powerful, since it provides the user with instantaneous keyboard control over the program. You can also use this function to advantage with subsequent IF-THEN statements, which can then be used for multiple-choice branching related to a specific keyboard input. Hitting any key moves the program to the GOSUB routine in lines 1700 through 1800, which prints the array and words and returns control to lines 1500 through 1600.

Line 1510 selects a word and assigns the string value D$. Line 1530 extracts the number of letters in the word which will be used in the counting sequence at line 1540. The error statement at line 1530 is a trap which catches a bad input at line 1550, for example, a keyboard bounce resulting in a coordinate outside the dimensions of the A$(X,Y) array.

The coordinates input at line 1550 are used to select the value of A$(X,Y), which is compared to the value of E$ derived from the MID$ function at line 1560. If the letters match, the A$(X,Y) value is changed to " ", providing positive feedback when the screen is reprinted.

Lines 1900 through 1940 provide the option to end the game or go back for another puzzle. Note the use of INKEY$ as a branching device. The DATA is stored starting at line 2000, and the error routine at line 3000.

Modifications and Changes

Everybody loves to play with a program; so I'll offer a few suggestions you may want to try. You can increase the number of words in the data bank to whatever number your available memory handles. This version of the program leaves about 9000 bytes available in a 16K machine; so there is plenty of space for expansion. To expand the data bank, you must change the C$(50) dimension at line 200 to match the number of words in the DATA file. Subsequently, line 210 has to CLEAR more string storage space, and lines 260 through 300 have to be modified to reflect the number of words available. You can insert more words into the puzzle by setting the value of A1 higher in lines 490, 590, 690, etc. and resetting the loop at line 1780. This modification increases the time required to place the words into the puzzle.

Program Listing. *Puzzler*

```
5 REM *** FIND IT  BY jAMES p. mORGAN, OCT 79 ***
10 CLS: PRINT@25,"FIND IT":PRINT
20 PRINT"WHILE YOU ARE READING THIS, THE COMPUTER IS BU
    SY
30 PRINT"CHURNING AWAY WITH THOUSANDS OF COMPUTATIONS W
    HICH
40 PRINT"WILL RESULT IN A WORD FINDER pUZZLE. iT TAKES
    ABOUT
50 PRINT"25 SECONDS TO SET UP THE PUZZLE...IT WILL BE C
    OMPLETE
60 PRINT"ANY SECOND NOW. yOU WILL BE SHOWN A SQUARE OF
    JUMBLED
70 PRINT"LETTERS IN WHICH EIGHT LISTED WORDS WILL BE HI
    DDEN.
80 PRINT"LOCATE A WORD, ENTER IT FROM THE KEYBOARD, AND
     THEN
90 PRINT"CONFIRM YOUR LOCATION BY ENTERING THE COORDINA
    TES--
100 PRINT"VERTICAL, THEN HORIZONTAL, OF EACH LETTER IN
    THE WORD.
110 PRINT"TO GET STARTED, TAP ANY KEY."
195 REM *** DIMENSION ARRAYS, POKE FOR READ EXECUTION *
    **
200 CLEAR 500: DIM A$(12,19), C$(50), W$(8):POKE16553,2
    55
205 REM *** SET MAIN ARRAY TO EMPTY SPACE ***
210 FOR X=1 TO 12
220 FOR Y=1 TO 19
230 A$(X,Y)="  "
240 NEXT Y
250 NEXT X
255 REM *** READ IN THE DATA FOUND AT LINE 2000 ***
260 FOR Z=1 TO 50
270 READ C$(Z)
280 NEXT Z
290 A1=0
295 REM *** SELECT WORD, ASSIGN TO W$, PULL LENGTH ***
300 Z=RND (50)
310 W$(A1)=C$(Z)
320 N=LEN(W$(A1))
325 REM *** SELECT INSERTION DIRECTION AT RANDOM ***
330 L=RND(8)
340 ON L GOTO 400,500,600,700,800,900,1000,1100
400 REM *** UP VERTICAL SELECT AND PLACE ***
410 X=RND(12): Y=RND(19)
420 FOR I=0 TO N-1
430   IF X-I=0 THEN 410
440   IF A$(X-I,Y)<>"  " THEN 410
450 NEXT I
460 FOR I=0 TO N-1
470   A$(X-I,Y)=MID$(W$(A1),I+1,1)
480 NEXT I: A1=A1+1
490   IF A1=8 THEN 1300 ELSE 300
500 REM *** DOWN VERTICAL SELECT AND PLACE ***
510 X=RND(12): Y=RND(19)
520 FOR I=0 TO N-1
530  IF X+I=13 THEN 510
540  IF A$(X+I,Y) <> "  " THEN 510
550 NEXT I
560 FOR I=0 TO N-1
570   A$(X+I,Y)=MID$(W$(A1),I+1,1)
580 NEXT I: A1=A1+1
590 IF A1=8 THEN 1300 ELSE 300
600 REM *** RIGHT HORZ. SELECT AND PLACE ***
610 X=RND(12): Y=RND(19)
620 FOR I=0 TO N-1
630   IF Y+I=20 THEN 610
640   IF A$(X,Y+I)<>"  " THEN 610
```

```
650 NEXT I
660 FOR I=0 TO N-1
670   A$(X,Y+I)=MID$(W$(A1),I+1,1)
680 NEXT I: A1=A1+1
690 IF A1=8 THEN 1300 ELSE 300
700 REM **LEFT HORZ. SELECT AND PLACE ***
710 X=RND(12): Y=RND(19)
720 FOR I=0 TO N-1
730   IF Y-I=0 THEN 710
740   IF A$(X,Y-I)<>"  " THEN 710
750 NEXT I
760 FOR I=0 TO N-1
770   A$(X,Y-I)=MID$(W$(A1),I+1,1)
780 NEXT I: A1=A1+1
790 IF A1=8 THEN 1300 ELSE 300
800 REM *** RIGHT UP D]AGONAL ***
810 X=RND(12): Y=RND(19)
820 FOR I=0 TO N-1
830   IF Y+I=20 THEN 810
840   IF X-I=0 THEN 810
850   IF A$(X-I,Y+I)<>"  " THEN 810
860 NEXT I
870 FOR I=0 TO N-1
880   A$(X-I,Y+I)=MID$(W$(A1),I+1,1)
890 NEXT I: A1=A1+1: IF A1=8THEN 1300 ELSE 300
900 REM *** LEFT DOWN DIAGONAL ***
910 X=RND(12): Y=RND(19)
920 FOR I=0 TO N-1
930   IF Y-I=0 THEN 910
940   IF X+I=13 THEN 910
950   IF A$(X+I,Y-I)<>"  " THEN 910
960 NEXT I
970 FOR I=0 TO N-1
980   A$(X+I,Y-I)=MID$(W$(A1),I+1,1)
990 NEXT I: A1=A1+1: IF A1=8 THEN 1300 ELSE 300
1000 REM *** LEFT UP DIAGONAL ***
1010 X=RND(12):Y=RND(19)
1020 FOR I=0 TO N-1
1030   IF X-I=0 THEN 1010
1040   IF Y-I=0 THEN 1010
1050   IF A$(X-I,Y-I)<>"  " THEN 1010
1060 NEXT I
1070 FOR I=0 TO N-1
1080   A$(X-I,Y-I)=MID$(W$(A1),I+1,1)
1090 NEXT I: A1=A1+1: IF A1=8 THEN 1300 ELSE 300
1100 REM *** RIGHT DOWN DIAGONAL ***
1110 X=RND(12): Y=RND(19)
1120 FOR I=0 TO N-1
1130   IF X+I=13 THEN 1110
1140   IF Y+I=20 THEN 1110
1150   IF A$(X+I,Y+I)<>"  " THEN 1110
1160 NEXT I
1170 FOR I=0 TO N-1
1180   A$(X+I,Y+I)=MID$(W$(A1),I+1,1)
1190 NEXT I: A1=A1+1: IF A1=8 THEN 1300 ELSE 300
1300 REM *** FILLING IN REST OF A$(X,Y) ARRAY ***
1310 FOR X=1 TO 12
1320 FOR Y=1 TO 19
1330   IF A$(X,Y) <>"  " THEN 1370
1340   B=RND(26)
1350   C=64+B
1360   A$(X,Y)=CHR$(C)
1370 NEXT Y
1380 NEXT X
1390 M$=INKEY$: IF M$="" THEN 1390
1400 GOSUB 1700
1500 REM *** MATCH UP ROUTINE ***
1510 INPUT"ENTER WORD. ENTER '*' TO END.";D$
1520 IF D$="*" THEN 1900
1530 N=LEN(D$):ON ERROR GOTO 3000
1540 FOR I=1 TO N
```

Program continued

```
1550   INPUT"ENTER VERTICAL,HORIZ. COORDINATES OF LETTE
     R MATCH.";X,Y
1560 E$=MID$(D$,I,1)
1570 IF E$<>A$(X,Y)THENPRINT"WRONG!! START OVER":FORX=1
     TO900:NEXT:GOTO10
1580 IF E$=A$(X,Y) THEN A$(X,Y)=" "
1590 GOSUB 1700
1600 NEXT I
1610 GOTO 1500
1700 REM *** THE SCREEN PRINT SUBROUTINE ***
1705 REM ***NEXT LINE, USE 2 SPACES UP TO 9, ONE SPACE
     AFTER ***
1710 CLS:PRINT"1  2  3  4  5  6  7  8  9 10 11 12 13 14
      15 16 17 18 19"
1720 FOR X=1 TO 12
1730 FOR Y=1 TO 19
1740 PRINTA$(X,Y);"  ";
1750 NEXT Y
1760 PRINTX
1770 NEXT X
1780 FOR A1=0 TO 8
1790 PRINTW$(A1);" ";
1800 NEXT A1: RETURN
1900 CLS:PRINT"IF YOU WISH TO PLAY ANOTHER ROUND, TYPE
     'Y' FOR YES."
1910 PRINT"TO END, JUST TAP THE SPACE BAR."
1920 G$=INKEY$: IF G$="" THEN 1920
1930 IF G$="Y"THEN 10
1940 PRINT:PRINT"GOODBY FOR NOW.": END
2000 DATA "COMPUTER","PAPER","TIGER","BYTE","BAUD","HOU
     SE"
2010 DATA "RANDOM","TABLE","GENERATOR","INPUT","OUTPUT"
     ,"OHM"
2020 DATA "RESISTOR","VIDEO","DISPLAY","KEYBOARD","LINE
     ","FOR"
2030 DATA "INPUT","INTERFACE","SIGNAL","MICRO","POWER",
     "SUPPLY"
2040 DATA "SYSTEM","RADIO","WATT","WAIT","STATE","DEVIC
     E"
2050 DATA "PROGRAM","BASIC","CASSETTE","LINEAR","TIMER"
     ,"BYTE"
2060 DATA "PLUS","MINUS","GAME","THEORY","HIGH","LOW","
     BUFFER"
2070 DATA "ELECTRON","AIR","FORCE","NUMBER","BINARY","H
     EX","END"
2995 REM *** ERROR ROUTINE NEXT ***
3000 CLS:PRINT"BAD INPUT. TRY AGAIN.":FORX=1TO1000:NEXT
3010 GOSUB 1700
3020 RESUME 1550
```

GRAPHICS

On Your Mark, Get Set, and Go

On Your Mark, Get Set, and Go

by Gerald DeConto

Computers are supposed to make our lives easier aren't they? Then why do we spend half our time either setting up or waiting for a graphics display? Since I couldn't answer this, I decided to do something about it; so here is a short program to draw (or erase) a line from one point on the screen to another. I call it GRAFX2.

GRAFX2 is a 255-byte extension of Level II BASIC. When it is in memory and the first section has been executed, it causes the computer to respond to the Disk BASIC commands LSET and RSET with something other than an L3 ERROR message. LSET is designed to construct a line from one point on the screen to another. RSET does the inverse and erases a line between two points. These two new commands are the descendants of SET and RESET and function in a very similar way. LSET and RSET, however, require the x-and y-coordinates for both the start and the end points. The other difference is that the coordinates must be integer variables. Using numbers or any other type of variable confuses GRAFX2, and it will not function correctly.

GRAFX2 plots about 1800 points a second. This means that you can clear the screen in about three-and-a-half seconds. Another feature of GRAFX2 is that it can easily be patched back into BASIC if the computer restarts (displays MEMORY SIZE?).

Program Concepts

Assume that you have just typed in a short BASIC program. By PEEKing into the program's memory locations you find characters and internal command codes. The codes are numbers ranging from 129 to 250. They are the one-byte representations of the BASIC commands you just typed in.

When you entered the BASIC text, an encoding routine was called automatically to reduce each reserved word to a one-byte code. This saves memory space and speeds up execution. If you list or edit the text, a decoding routine displays the commands in their original, readable form.

Since you have a program, you can use the RUN command. When the RUN command is used, it causes a series of machine-language instructions to be performed in ROM. It is responsible for stepping through BASIC programs and executing any command codes (commands) it finds. The computer does this until you stop it or instruct it to do otherwise.

To execute a command you must locate the position of the command code in the command code list. Once the location is found, a second list containing memory address pointers is scanned to obtain the corresponding entry. For most commands, this address points to a subroutine in ROM that does what the command was defined to do. The computer jumps (or calls) to this address.

There are exceptions to the case. The Disk BASIC commands, for instance, have their pointers send the computer to the reserved areas in RAM, where a series of JP instructions is found. The computer is sent to the correct JP, and the JP sends it to the L3 ERROR routine if there is no disk. If a disk is present, the computer goes to the appropriate disk operating system routine. The important thing to remember is that these JP instructions are in RAM. This means that you can alter them to send the computer to the right entry point for each command in GRAFX2.

The Technical Aspect

The first 23 bytes of GRAFX2 change the JP instructions for the disk commands LSET and RSET, giving you an easy way out of BASIC. Level II BASIC uses the HL register to keep tabs on the current position in a program. Once a command code is found, HL points to the very next character in the text—in this case, the first bracket. In order to move HL past this and skip any spaces in the text, use RST 10H. The latter routine reads in the name of the variable HL pointed to and stores the value of this variable at 4121H. Note that this applies only to integer variables, since 2540H stores other variables at different locations and in different formats. Numbers are not read in by 2540H. GETVAL is called four times to get data. It also checks to see that the values fall within the screen's limits. If they do not, then the computer goes to the L3 ERROR routine at 12DH.

In developing the program, I recalled what I knew about graphing lines. I needed to know the difference between the y-coordinates (rise) and the difference between the x-coordinates (run). These are easy to calculate, but I also needed the slope. The slope is the slant of the line; it's found by dividing the rise by the run. This involves the use of division and good knowledge of floating point numbers. I also needed to know how to multiply in machine language, since, to get a new y-coordinate, I had to multiply the slope by the next x-coordinate. This is easy to do in BASIC but quite hard and slow in machine language.

I solved my problems by remembering that division can be simulated by successive subtraction, and multiplication can be simulated by successive addition. With a little bit of brain work I figured out a way to do what I needed, and it used only integers. I ended up with a running tally which is similar to the numerator of the slope. It keeps tabs on when I should increment and which coordinate to use.

This leads you to the Level II graphics routine at 150H. This routine gets its data from the stack, with the exception of the y-coordinate, which is left in the A register before the program jumps to 150H. You must establish the return address, the set or reset code, and the x-coordinate before going to 150H.

One thing I should mention about this routine is that it requires an ADD HL,HL instruction (29H) at the return address for some of its calculations. Since this doesn't harm anything in this program, you can safely ignore it.

When you return from setting or resetting the current point, you must test the current point and the end point for equality. If they are equal, then you can go back to BASIC. You calculate the new x- and y-coordinates if the points are not equal.

Remember that before loading in the program, you must set the memory size at about 32250 to avoid having the BASIC stack wipe out any part of GRAFX2. Note that I did not place this program at the very top of memory, because I had my serial printer driver at the top. As I said before, the commands need four data values—an x and a y for both the start and end points. They must be integer variables for GRAFX2 to work. If the computer restarts, you can restart the program by entering SYSTEM and then /32256. This puts your memory pointers in reserved RAM to link GRAFX2 into BASIC.

This program is very useful in all types of computing. It draws in any direction and is very good at doing borders, histograms, charts, or whatever you need. You can also use it as a module for some of your own programs.

Program Listing. *GRAFX2*

Encyclopedia Loader

```
               ;*************************************************
               ;         LEVEL II BASIC COMMAND EXTENSION
               ;*************************************************
               ;         NAME:    GRAFX2
               ;         BY   :   G. DECONTO
               ;                  1415 SOUTH LAKESIDE DR.
               ;                  WILLIAMS LAKE, BRITISH COLUMBIA
               ;                  CANADA, V2G 3A7
               ;
               ;         NEW COMMANDS:
               ;            LSET(W,X,Y,Z) AND RSET(S,T,U,V)
               ;
               ;    LSET DRAWS A LINE FROM ONE POINT (START) TO
               ;    ANOTHER (END).
               ;    RSET IS ITS INVERSE AND ERASES A LINE.
               ;
               ;    BOTH START AND END POINTS ARE DEFINED BY ONE
               ;    X AND ONE Y COORDINATE.THESE COORDINATES MUST
               ;    BE INTEGER VARIABLES FOR GRAFX2 TO WORK.
               ;    MEMORY MUST BE RESERVED AT ABOUT 32255 TO
               ;    AVOID WIPING OUT GRAFX2.
               ;
               ;    IN CASE OF COMPUTER RESTART ( GOES "MEMORY
               ;     SIZE?") JUST ENTER "SYSTEM" AND "/32256"
               ;    AT THE PROMPT.
               ;*************************************************
7EØØ                     ORG     7EØØH
7EØØ 3EC3      START     LD      A,ØC3H             ;"JP" CODE
7EØ2 21177E              LD      HL,LSET            ;ADDRESS OF LSET
7EØ5 329741              LD      (4197H),A
7EØ8 229841              LD      (4198H),HL
7EØB 211B7E              LD      HL,RSET            ;ADDRESS OF RSET
7EØE 329A41              LD      (419AH),A
7E11 229B41              LD      (419BH),HL
7E14 C3CCØ6              JP      6CCH               ;RETURN TO BASIC
7E17 3E8Ø      LSET      LD      A,8ØH              ;CODE FOR "SET"
7E19 18Ø2                JR      CONT
7E1B 3EØ1      RSET      LD      A,Ø1H              ;CODE FOR "RESET"
               ;=================================================
               ;         HERE ON COMMANDS MERGE AND ARE IDENTICAL
               ;         HL POINTS TO CHARACTER AFTER COMMAND
               ;=================================================
7E1D C5        CONT      PUSH    BC                 ;SAVE BC
7E1E D5                  PUSH    DE                 ;SAVE DE
7E1F 32FD7E              LD      (WHICH),A          ;(WHICH)=SET/RSET
               ;=================================================
               ;         GET X1,Y1,X2 AND Y2.TEST FOR ERRORS.
               ;=================================================
7E22 3E8Ø                LD      A,8ØH
7E24 32FE7E              LD      (LIMIT),A          ;X LIMIT
7E27 CDE27E              CALL    GETVAL             ;GET X1
7E2A 32F47E              LD      (X1),A
7E2D 3E3Ø                LD      A,3ØH
7E2F 32FE7E              LD      (LIMIT),A          ;Y LIMIT
7E32 CDE27E              CALL    GETVAL             ;GET Y1
7E35 32F57E              LD      (Y1),A
7E38 3E8Ø                LD      A,8ØH
7E3A 32FE7E              LD      (LIMIT),A          ;X LIMIT
7E3D CDE27E              CALL    GETVAL             ;GET X2
7E4Ø 32F67E              LD      (X2),A
7E43 3E3Ø                LD      A,3ØH
7E45 32FE7E              LD      (LIMIT),A          ;Y LIMIT
7E48 CDE27E              CALL    GETVAL             ;GET Y2
7E4B 32F77E              LD      (Y2),A
7E4E 23                  INC     HL                 ;IGNORE BRACKET
7E4F E5                  PUSH    HL                 ;SAVE HL
               ;=================================================
               ;         CALCULATE RISE,RUN AND THE INCREMENTS
```

```
                    ;==============================================
7E50 21F47E         GETRUN  LD      HL,X1           ;X VARIABLES
7E53 CDBE7E                 CALL    RYZRUN
7E56 21F57E         GETRYZ  LD      HL,Y1           ;Y VARIABLES
7E59 CDBE7E                 CALL    RYZRUN
7E5C 3AFB7E                 LD      A,(RISE)
7E5F 32FC7E                 LD      (NUMER),A       ;NUMERATOR = RISE
                    ;==============================================
                    ;       GET READY,JUMP TO ROM SET/RESET ROUTINE
                    ;==============================================
7E62 21747E         LOOP    LD      HL,RETADD       ;RETURN ADDRESS
7E65 E5                     PUSH    HL
7E66 3AFD7E                 LD      A,(WHICH)       ;GET CODE
7E69 F5                     PUSH    AF
7E6A 3AF47E                 LD      A,(X1)          ;GET CURRENT X
7E6D F5                     PUSH    AF
7E6E 3AF57E                 LD      A,(Y1)          ;GET CURRENT Y
7E71 C35001                 JP      150H            ;GOTO GRAPHICS
7E74 29             RETADD  ADD     HL,HL           ;29H THIS ADDRESS
                    ;                               AVOIDS "SN ERROR"
                    ;==============================================
                    ;       TEST TO SEE IF WE ARE DONE
                    ;==============================================
7E75 3AF67E         TEST    LD      A,(X2)
7E78 47                     LD      B,A
7E79 3AF47E                 LD      A,(X1)
7E7C B8                     CP      B               ;X1=X2 ?
7E7D 200E                   JR      NZ,NXTTST
7E7F 3AF77E                 LD      A,(Y2)
7E82 47                     LD      B,A
7E83 3AF57E                 LD      A,(Y1)
7E86 B8                     CP      B               ;Y1=Y2 ?
7E87 2004                   JR      NZ,NXTTST
7E89 E1                     POP     HL              ;POINTS ARE EQUAL
7E8A D1                     POP     DE              ;SO RESTORE THE
7E8B C1                     POP     BC              ;REGISTERS.
7E8C C9                     RET                     ;GO HOME!
                    ;==============================================
                    ;       NOT DONE, DO WE INCREMENT X OR Y ?
                    ;==============================================
7E8D 3AFA7E         NXTTST  LD      A,(RUN)
7E90 47                     LD      B,A
7E91 3AFC7E                 LD      A,(NUMER)
7E94 B8                     CP      B       ;NUMERATOR < RUN ?
7E95 3812                   JR      C,INCX  ;JUMP IF YES
7E97 B7             INCY    OR      A       ;NUMERATOR >= RUN
7E98 98                     SBC     A,B     ;NUMERATOR=NUMERATOR-RUN
7E99 32FC7E                 LD      (NUMER),A
7E9C 3AF97E                 LD      A,(YINC)
7E9F 47                     LD      B,A
7EA0 3AF57E                 LD      A,(Y1)
7EA3 80                     ADD     A,B             ;INCREMENT Y
7EA4 32F57E                 LD      (Y1),A
7EA7 18B9                   JR      LOOP            ;KEEP GOING
7EA9 47             INCX    LD      B,A
7EAA 3AFB7E                 LD      A,(RISE)
7EAD 80                     ADD     A,B     ;NUMERATOR=NUMERATOR+RISE
7EAE 32FC7E                 LD      (NUMER),A
7EB1 3AF87E                 LD      A,(XINC)
7EB4 47                     LD      B,A
7EB5 3AF47E                 LD      A,(X1)
7EB8 80                     ADD     A,B             ;INCREMENT X
7EB9 32F47E                 LD      (X1),A
7EBC 18A4                   JR      LOOP            ;KEEP GOING
                    ;==============================================
                    ;       SUBROUTINE TO GET RISE OR RUN
                    ;       USES BRACKET CONTENTS DURING SECOND CALL
                    ;==============================================
7EBE 46             RYZRUN  LD      B,(HL)          ;GET X1 (Y1)
7EBF 110200                 LD      DE,2
7EC2 19                     ADD     HL,DE           ;POINT TO X2 (Y2)
```

Program continued

```
7EC3 7E         Ø149Ø        LD      A,(HL)          ;GET X2 (Y2)
7EC4 4F         Ø15ØØ        LD      C,A
7EC5 B8         Ø151Ø        CP      B
7EC6 28Ø6       Ø152Ø        JR      Z,EQUAL         ;X2=X1 (Y2=Y1)
7EC8 38Ø8       Ø153Ø        JR      C,LESS          ;X2<X1 (Y2<Y1)
7ECA 3EØ1       Ø154Ø GREATR LD      A,1      ;X2>X1 (Y2>Y1),INCREM.=1
7ECC 18ØC       Ø155Ø        JR      CONT2
7ECE 3EØØ       Ø156Ø EQUAL  LD      A,Ø             ;INCREMENT=Ø
7EDØ 18Ø8       Ø157Ø        JR      CONT2
7ED2 3EFF       Ø158Ø LESS   LD      A,ØFFH          ;INCREMENT=-1
7ED4 19         Ø159Ø        ADD     HL,DE           ;PT TO INCREMENT
                Ø16ØØ ;                              STORAGE
7ED5 77         Ø161Ø        LD      (HL),A
7ED6 78         Ø162Ø        LD      A,B             ;EXCHANGE X1 (Y1)
                Ø163Ø ;                               AND X2 (Y2)
7ED7 41         Ø164Ø        LD      B,C
7ED8 18Ø3       Ø165Ø        JR      FIGURE
7EDA 19         Ø166Ø CONT2  ADD     HL,DE           ;PT TO INCREMENT
                Ø167Ø ;                               STORAGE
7EDB 77         Ø168Ø        LD      (HL),A
7EDC 79         Ø169Ø        LD      A,C             ;RESTORE X2(Y2)
7EDD 19         Ø17ØØ FIGURE ADD     HL,DE           ;PT TO RUN (RISE)
7EDE B7         Ø171Ø        OR      A               ;CLEAR CARRY
7EDF 98         Ø172Ø        SBC     A,B             ;RUN=X2-X1
                Ø173Ø ;                               (RISE=Y2-Y1)
7EEØ 77         Ø174Ø        LD      (HL),A
7EE1 C9         Ø175Ø        RET
                Ø176Ø ;================================================
                Ø177Ø ;      SUBROUTINE TO GET VALUES FROM VARIABLES
                Ø178Ø ;================================================
7EE2 D7         Ø179Ø GETVAL RST     1ØH             ;SKIP BLANKS
7EE3 CD4Ø25     Ø18ØØ        CALL    254ØH           ;PUT VARIABLES
                Ø181Ø ;                               VALUE AT 4121H
7EE6 3AFE7E     Ø182Ø        LD      A,(LIMIT)
7EE9 47         Ø183Ø        LD      B,A
7EEA 3A2141     Ø184Ø        LD      A,(4121H)       ;GET VALUE
7EED B8         Ø185Ø        CP      B               ;WITHIN LIMITS ?
7EEE D8         Ø186Ø        RET     C               ;IF YES,RETURN
7EEF E1         Ø187Ø L3ERR  POP     HL
7EFØ E1         Ø188Ø        POP     HL
7EF1 C32DØ1     Ø189Ø        JP      Ø12DH           ;L3 ERROR ROUTINE
7EF4 ØØ         Ø19ØØ X1     DEFB    Ø
7EF5 ØØ         Ø191Ø Y1     DEFB    Ø
7EF6 ØØ         Ø192Ø X2     DEFB    Ø
7EF7 ØØ         Ø193Ø Y2     DEFB    Ø
7EF8 ØØ         Ø194Ø XINC   DEFB    Ø
7EF9 ØØ         Ø195Ø YINC   DEFB    Ø
7EFA ØØ         Ø196Ø RUN    DEFB    Ø
7EFB ØØ         Ø197Ø RISE   DEFB    Ø
7EFC ØØ         Ø198Ø NUMER  DEFB    Ø
7EFD ØØ         Ø199Ø WHICH  DEFB    Ø
7EFE ØØ         Ø2ØØØ LIMIT  DEFB    Ø
7EØØ            Ø2Ø1Ø        END     START
ØØØØØ TOTAL ERRORS
```

HARDWARE

Program an EPROM

HARDWARE

Program an EPROM

by Dr. Steven A. Larson N3SL

Now that the prices of EPROMs are dropping to more tolerable levels for hobbyists, the potential for adding one or more of these little gems to the TRS-80 is within most everyone's reach. First, I will describe a method of adding 2000 bytes of EPROM programs for about $15. The second part of this chapter describes an inexpensive (under $15) programmer designed specifically for the 2048-byte, single 5-volt version 2716 EPROM, using wire-wrap technique and easy-to-find parts.

An Additional 2000 Bytes of Permanent Memory

How would you like to have 2000 additional bytes of ROM routines of your choosing—or of your own design? It takes only three common integrated circuits, one EPROM, a small 5-volt power supply, and a cable to attach either to your expansion interface or directly to your expansion port. The main component in the unit is the 2048-byte, single 5-volt supply, 2716 EPROM, which costs as little as $5. The total cost, even for new parts, is only about $15 if you shop around. Currently there are several commercial units available that are similar to the EPROM I will describe here, but prices start at about $70.

If you look at the memory map in the back of the *Level II BASIC Reference Manual*, you will notice that just above the 12K of BASIC ROM is an area from 3000 to 37DD (hex) that is unused. Exatron uses this area for their Stringy-Floppy unit, and there's nothing to stop us from using that same area—unless you already have an Exatron unit, but even that can be taken care of with a simple switch.

I first began thinking about this project when I realized there were several programs that I kept using over and over. Before I had disk drives and the new keyboard driver in DOS, KBFIX was an absolute necessity for any work of more than one minute in front of the keyboard; so when the idea of permanently available programs came up, KBFIX was an obvious choice. Secondly, again before my disk era, I spent a lot of time making—or trying to make—backup copies of my tapes, and I used a program called TRCOPY almost daily. In its original form it was only about 1800 bytes long, so I knew it would fit quite nicely into a 2K EPROM (but with substantial address modifications). Finally, after adding my upper/lowercase character generator, loading the driver to it became a hassle, so that too was chosen to go into the EPROM. Now with my EPROM board attached, I can use a mere SYSTEM command, and I have any or all of the utilities immediately available, with no possibility of loading errors, no memory size to set, and no chance of wiping them out with DOS.

The scenario wasn't entirely as easy as I've made it sound, however, as I knew no one other than commercial firms who could program my EPROMs—at a reasonable price; so I set out to design my own EPROM programmer for the 5-volt version of the 2716 and eventually wound up with a nine-IC unit that I wire-wrapped for a grand total of $12.

The Circuit

The circuit for the new memory card is quite simple, containing only four integrated circuits including the EPROM. The last 49 bytes of the EPROM are unusable due to address decoding limitations of the three ICs used. You will notice from the memory map, however, that the TRS-80 picks up the addresses again at 37DEH; so the last 34 bytes of the EPROM block—from 37DEH to 37FFH—aren't available for our use anyway. Thus, the real loss in usable EPROM space is only 15 bytes. In order to decode all the way to 37DDH to gain those 15 bytes back, I figured two additional ICs would be needed, and that wasn't worth the money. The schematic is shown in Figure 1. The three integrated circuits U1–3 are used to decode the address being sent out by the CPU.

Address Decoding

The addresses for this EPROM range from 3000H to 37CFH, and the three ICs cover that entire range. Decoding A12–15 is straightforward. These lines are only valid for the EPROM when they equal a binary 3, i.e., when A15 and A14 are low (0s), and A13 and A12 are high (1s). Decoding is done by U1A and U2A as follows: The output of U1A, an OR gate, is low only if both A14 and A15 are low. The output of U2A, a NAND gate, is low only if both A12 and A13 are high. Thus, when the CPU is sending out any address in the 3000 hex block, the outputs of both U1A and U2A are low. Since we will actually need an active high for final decoding (through U3C), the low from lines A14 and A15 is immediately fed into U2B, which is wired as an inverter.

Next, since we never need an address of 3800H or greater, line A11, which determines the 8 or greater part, was brought into the decoding directly at pin 5 of U1. Here it is ORed with the previously determined 3 so that the output of U1B is low only if the most significant byte of the address being sent is in the range of 30–37H—in other words, 3800H and above are not decoded. Again, since we actually need a high in the final decoding, this low is fed into U2C which is wired as an inverter and then into U3C, the last decoding gate.

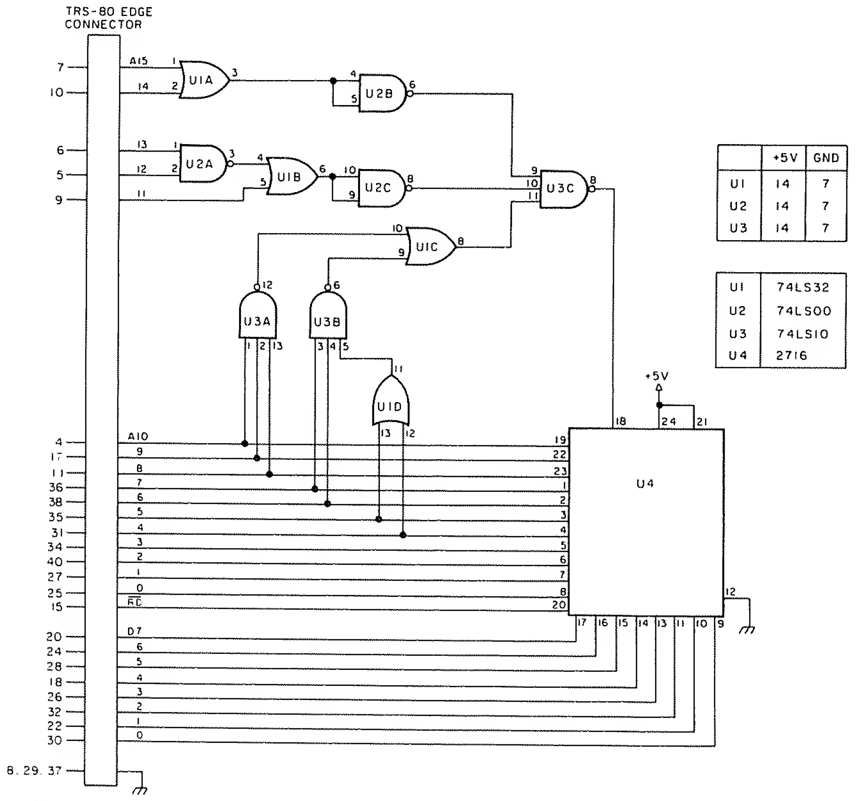
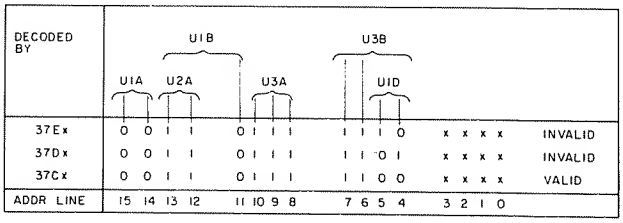


Figure 1. *EPROM memory board schematic diagram. Numbers to the left of the edge connector block indicate TRS-80 bus pin designations.*

Finally, things get a little more complicated as we determine if the address is 37CFH or lower—the only valid addresses for our EPROM. It is easiest to see this by looking at the actual binary addresses in question, as shown in Figure 2. Since the 30H to 37H part of the address has already



DECODED BY			U1B		U1B				U3B	U3B	U3B						
	U1A	U1A	U2A	U2A		U3A	U3A	U3A			U1D	U1D					
37Ex	0	0	1	1	0	1	1	1	1	1	1	0	x	x	x	x	INVALID
37Dx	0	0	1	1	0	1	1	1	1	1	0	1	x	x	x	x	INVALID
37Cx	0	0	1	1	0	1	1	1	1	1	0	0	x	x	x	x	VALID
ADDR LINE	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	

Figure 2. *Memory board address decoding scheme. If either A4 or A5 is high while both A6 and A7 are high, the address is invalid, as indicated.*

been decoded, the only thing left to determine is whether the least significant byte of the address is CF or less, which we do in a roundabout way by determining if the byte is D0 or greater. By looking at Figure 2, you can see that the key lies in decoding lines A4–7. If both or either A4 or A5 is high at the same time that A6 and A7 are both high, the hex value must be D0 or greater. We obviously can't disallow all D0 or greater states, so we need to determine when x7D0 or greater exists and disallow that situation only. This is done by U3A, U3B, and U1D. Whenever addresses 3700–37FFH are output, all three inputs to U3A (A8–10) are high from the 7, and its output is low. Looking at Figure 1 again, notice that whenever addresses xxD0 to xxFF are output, either or both A4 and A5 are high, setting the output of U1D high. Combining that output with A6 and A7 at the input of U3B will give a low output there whenever the D0 or greater state is on the address bus. By ORing the outputs of U3A and U3B together in U1C, the output of U1C will be high at all times except when x7D0 to x7FFH is on the address bus, and these are the addresses we wanted to exclude.

The final decoding is done in U3C, which simply ANDs what we've just been through. When (1) A15 and A14 are low *and* (2) A13 and A12 are high and A11 is low *and* (3) x7D0–x7FFH is not on the address bus, then a low pulse is output from U3C pin 8. This low is fed into the chip enable pin on the EPROM which selects it for the read cycle in progress.

The only other pin on the EPROM that needs special attention is the output enable (pin 20) which, when low, takes the data lines from a high impedance state and connects them to the data bus during the read pulse from the CPU.

Construction

Circuit layout is not critical and wire-wrap technique can be used. Designing a printed circuit board would be fairly difficult and would require either double-sided boards or the extensive use of jumpers. I figured that wire-wrappping four ICs, even though it looks messy, was far easier than laying out a printed circuit board. The power supply can be small. If you use 74LS series integrated circuits, the current drawn during an EPROM access is only about 120 mA, and when no EPROM read is in progress it drops to about 45 mA due to a standby feature of the EPROM. I used a small, 5-volt wall power pack like those supplied with calculators or games and have not had any problems with heat or erratic behavior in the circuit.

How to Use It

Use of the new memory is as simple as typing SYSTEM and entering the appropriate address. Since the three programs I mentioned did not fill up my EPROM, I spread them out a little so that the decimal addresses were easier to remember. Since I needed KBFIX every time I used the computer, I programmed it at 3039H, which is 12345 decimal. TRCOPY was programmed at 12500 decimal; 14000 activates my lowercase driver, and 14100 deactivates it. Obviously, the locations of your routines are up to you.

If you want to add more programs than a single EPROM will allow, you can have two or more EPROMs in the circuit. The modifications are minimal, involving simply wiring additional EPROM sockets in parallel with the first and running the chip enable line (U3B pin 8) through a switch to the appropriate EPROM. Be sure your power supply can handle the additional current each EPROM will add (about 25 mA).

The potential for a whole bank of useful utilities, subroutines, or even BASIC programs is limited only by the 2000-byte confine of the circuit. Within that range, however, the only factor would be your own imagination.

Build a 2716 EPROM Programmer for Under $15

The biggest obstacle in adding an EPROM to your system is getting the device programmed. I am going to describe an inexpensive programmer designed specifically for the 2048-byte, single 5-volt version 2716 EPROM. I purposely will go into fairly technical detail of the hows and whys of the circuit design, so that you can modify my circuit or completely redesign it to fit your own needs. As you will see, there are many ways to accomplish the task and many different integrated circuits that can be used.

Background

As I mentioned previously, the memory map in the back of the *Level II BASIC Reference Manual* shows that immediately above BASIC ROM there is a block of 2013 bytes from 3000H to 37DDH that is unused. Once I decided to fill this gap with an EPROM, I needed an inexpensive method of programming it. Buying a commercially-made programmer was financially out of the question. Of all the TRS-80 users I knew, nobody had an EPROM programmer. The lowest price I found at a computer dealer was $20 per EPROM, so the obvious alternative was to design my own. I needed something that was going to be easy to build, used commonly available parts, and could be interfaced directly to the TRS-80. The unit presented here fills all those criteria and has worked without a flaw on almost 75 programmed EPROMs.

The 2716 EPROM

At the time of this writing, the single 5-volt supply 2716 EPROM was being sold for as low as $5. Some confusion may result because different manufacturers use different part numbers, but in general the ads will specify a single voltage version if the chip is listed as a 2716. An equivalent chip is the TMS2516, and to my knowledge, that number is unique to the 5-volt type. Throughout this chapter I will use the number 2716, and I will be referring to the single-supply type exclusively.

The pin configuration of the 2716 is shown in Figure 3. The eleven address lines A0–A10 and the eight data lines D0–D7 are straightforward and directly correspond to their counterparts on the TRS-80 address and data buses. Pin 12 is a ground and needs no explanation. Power requirement is the standard +5-volt level of most TTL ICs and is applied to pin 24, labeled V_{CC}. Pin 21, labeled V_{PP} (programming voltage), has +5 volts applied to it as well, except during the time the EPROM is actually being programmed. Then, V_{PP} must have +25 volts applied to it. Pin 20 is the $\overline{OE}$, or output enable line, which is active low and will take the data lines from their normal high impedance state and gate them directly onto the data bus during a read cycle. Pin 18 is the $\overline{CE}$, or chip enable, serving as the power control line, which, when brought low, will select the EPROM for access by the CPU. It serves a dual purpose: When the EPROM is in the programming mode, a high applied to pin 18 for 50–55 ms serves as the programming pulse, hence the label $\overline{CE}$/PGM in Figure 3.

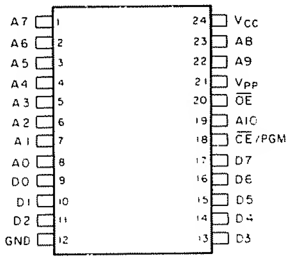


Figure 3. *Pin configuration for the Intel 2716 EPROM or any equivalent.*

We will not be concerned with the read cycle of the 2716 here, as the programmer presented does not have that capability. However, in order to understand the circuit design and the actual programming steps, details of the write cycle will be discussed.

The EPROM is in the programming mode when V_{PP}, pin 21, is at +25 V, and pin 20, the $\overline{OE}$ line, is high. When the address and data lines are stable, a 50–55 ms +5-volt pulse applied to pin 18 will write the data byte into the currently addressed location. In order to do this under TRS-80 software control, however, some means of keeping the data and address lines stable for this length of time has to be provided. Fifty milliseconds may not seem long, but in that time interval the TRS-80 will execute several thousand instructions, each potentially involving different addresses and data; so we cannot directly connect our EPROM lines to the TRS-80 address and data buses, as they are changing every microsecond. This problem is solved by the use of data latches, integrated circuits that can input data and hold—or latch—it indefinitely when properly clocked to do so. By routing the computer's address and data lines to the inputs of such latches and providing a latching pulse, the latches will hold the address and data bytes stable for the 50–55 ms period the EPROM requires. After that we are free to send a new address and a new data byte to the latches for the next cycle.

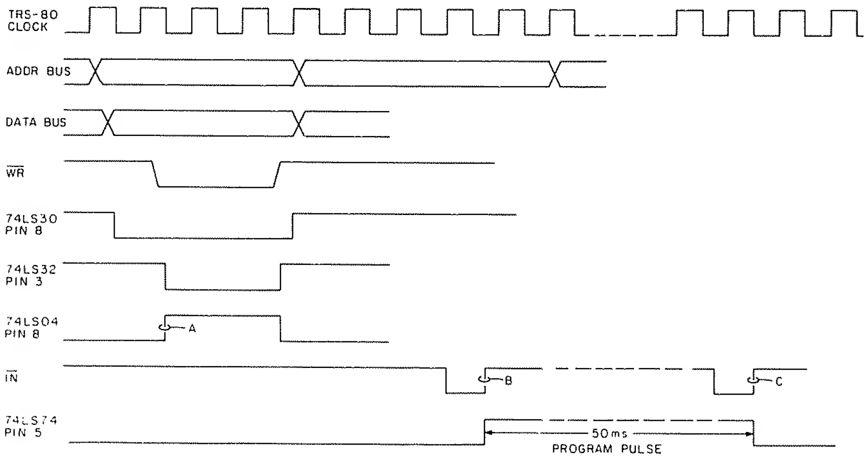


Figure 4. *Circuit timing diagram. Minor propagation time delays are not indicated, as they are not significant in this case.*

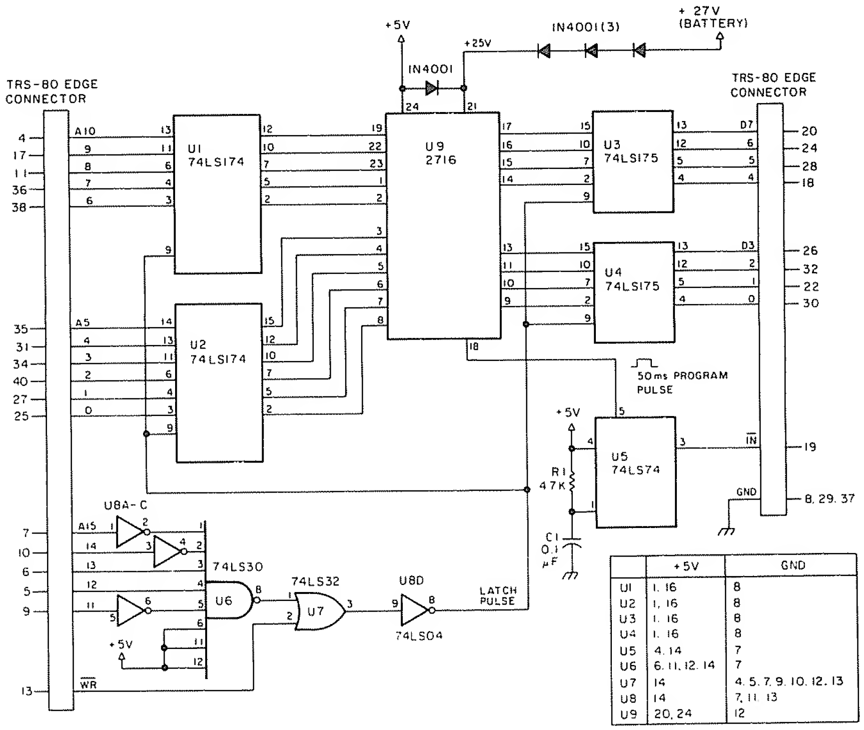
Circuit Design

Circuit design was planned to use easily obtainable, inexpensive parts, while minimizing part count. Other ICs could have been used and possibilities will be discussed later. The timing diagram for the circuit is shown in Figure 4, and the schematic in Figure 5.

Referring to Figure 5, U1 and U2 are 74LS174 hex D flip-flop ICs that serve to latch the 11 address lines we use. Whatever data is present at the input of the flip-flop when a positive-going pulse is applied to the clock input, pin 9, will be transferred to the output and held there until the next positive-going clock pulse is received. This pulse is represented by point A in Figure 4 and is generated whenever the CPU executes an LD (xxxx), A command, where xxxx is any address in the EPROM's range of 3000H to 37FFH. Likewise, U3 and U4, 74LS175 quad D flip-flops, latch the eight bits from the data bus when the same low-to-high pulse is applied to their clock input, pin 9. Thus, we have the means for providing stable address and data inputs to the EPROM during the write cycle.

To obtain the clock pulse needed for the latches, we have to use a method that limits the pulse to only those times when we actually want to program the EPROM; so simply using the WR line from the CPU or any similar line will not work. Since the EPROM has to occupy the memory space from 3000H to 37CFH, it seemed only logical that it should be addressed there, during programming. In order to accomplish this objective, the addresses from 3000H to 37FFH have to be decoded. In reality only the 3 needs to be decoded, since whenever the CPU is outputting an address in the 3000H range during the program, it must be addressing the EPROM by design.

Address decoding is done very simply by U8, a 74LS04 hex inverter,



	+5V	GND
U1	1, 16	8
U2	1, 16	8
U3	1, 16	8
U4	1, 16	8
U5	4, 14	7
U6	6, 11, 12, 14	7
U7	14	4, 5, 7, 9, 10, 12, 13
U8	14	7, 11, 13
U9	20, 24	12

Figure 5. *Circuit schematic. Numbers outside the edge connector block indicate TRS-80 bus pin numbers.*

and U6, a 74LS30 eight-input NAND gate. By including address line A11 in the decoding scheme, we are actually decoding 30xx through 37xx rather than the entire 3000H block, and a low is output from U6 whenever the EPROM board is being addressed. This low is combined with the $\overline{\text{WR}}$ pulse from the computer such that, whenever both are low (active) at the same time, the output of the OR gate, U7, also will be low. Now all that has to be done is a simple inversion through gate U8D, and we have the positive clock pulse we need to strobe the address and data into the appropriate latches.

Next, we must provide the actual 50–55 ms programming pulse and apply it to pin 18 of the EPROM. There are several ways of generating this pulse, but to me the simplest is to let the computer do the work. U5, a 74LS74, is our old friend the D-type flip-flop again. In this case, though, it is wired so that every time it receives a clock pulse, the state of pin 5, the Q output, will simply toggle from high-to-low or low-to-high. By placing the clock pulse under direct control of the TRS-80, we can control the length of time pin 5 is in a high state—specifically, 50–55 ms for our programming pulse. The clock pulse has to originate from a line from the CPU that is otherwise not used in our program's operation. One possibility that fits this restriction is the IN line, which normally is active only during cassette operations. Whenever an INput command (in machine language) is executed by the CPU, the IN line goes low for approximately 180 nanoseconds. We can use this pulse to cause U5 to toggle as just described. The timing sequence is shown in Figure 4, points B and C. You will again notice that it is the low-to-high transition that affects the actual clocking. The program steps that bring this about will be discussed later.

The only other components to note on the schematic, Figure 5, are R1 and C1, both associated with U5. Since we must have a high level output from U5 for our programming pulse, the state of the flip-flop has to be known before we start. Otherwise the programming loop could be entered with the flip-flop in the high state already, and we would toggle it to the low state for 50 ms. Obviously, that would not program the EPROM. Without R1–C1 in the circuit, the Q output of U5 will be randomly high or low when power is applied. The combination of R1–C1 assures that the flip-flop will initialize with pin 5 low. For normal operation of the flip-flop, both pin 4, the SET, and pin 1, the RESET or CLEAR, are tied to +5 volts. Bringing pin 1 momentarily low will clear the flip-flop to a state where pin 5 is low, which is what we need. When power is applied to the circuit, C1 is discharged and appears as a direct short to ground for pin 1, thus clearing the flip-flop and forcing pin 5 low. However, C1 rapidly charges, blocking the flow of dc current and effectively lifting pin 1 from ground. At that point, pin 1 is essentially at +5 volts through R1, where it remains until power is removed.

Power supply requirements for the circuit are minimal. Any good + 5-volt dc source capable of supplying approximately 250 mA will suffice. The programming voltage is obtained from three 9-volt batteries in series and from dropping the resulting 27 volts to the required 25 volts through 3 diodes, as shown in the schematic. (Each diode provides a 0.7-volt drop.) Any other 25-volt dc source should work as well.

Construction

The number of possible alterations to the circuit and the choice of ICs are quite large, but whatever you decide to use should follow the outlined principles. For example, an 8212 8-bit I/O port could be used as a data latch, as could a 74LS374 octal D flip-flop. I avoided these because of the expense, not only of the chips, but of the larger sockets they require. Likewise, the programming pulse could have been obtained using a 74LS121 or 74LS123 wired as a one-shot, but this would require timing resistors and capacitors and would have added unnecessary expense and design headaches.

Evidently the actual layout of the circuit is not at all critical, as I used wire-wrap technique without any difficulties. Designing a printed circuit board for a more permanent unit probably would not be difficult either. Interconnection to the TRS-80 expansion interface or expansion port should be through a ribbon cable with appropriate connectors and should be kept as short as possible. The cable in Photo 1 is a home-brew affair that has worked for many different breadboard projects over the last two years and works fine with the EPROM programmer as well as the EPROM memory board described in this chapter.

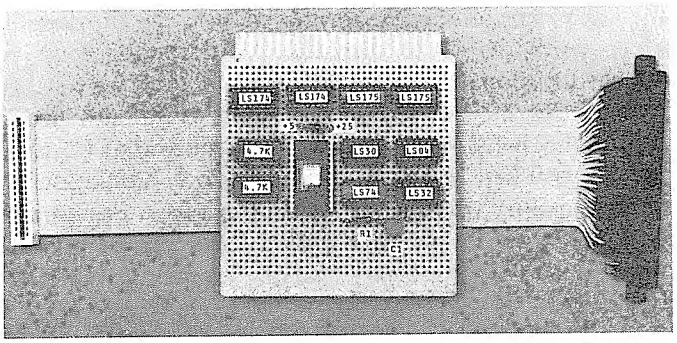


Photo 1. *Finished EPROM programmer board using wire-wrap technique. The home-brew cable is also shown. The 25- and 5-volt supply lines are brought into the indicated positions with clip leads.*

Programming

The Program Listing, which I will refer to as Burn-it, is an example of the steps necessary to program the EPROM. Burn-it can be located wherever you have room in memory. As written, it expects to find the program to be loaded into the EPROM stored between 9000H and 97CFH. I chose that location so that checking or modifying the program would be relatively easy, since all addresses would be displaced by 6000H and therefore easy to verify. You can change this to fit your particular memory size. Once the EPROM program is loaded and confirmed, you are ready to jump to Burn-it. For the most part, the listing is self-explanatory. When the program gets to the first IN A,(0) command, it is ready to generate the programming pulse previously described. The IN A,(0) command serves no purpose other than providing a pulse on the IN line which toggles U5 to a high state. The program delays for approximately 51.3 ms by counting down from 3500 to zero using a ROM routine at address 60H and executes another IN command which toggles U5 back to a low state, turning off the programming pulse. The CALL to ROM address 22CH was added after my first EPROM run. Programming 2000 bytes at 50 ms per byte takes 100 seconds. That was an unbearably long time to wait; so I added the call to provide a visual indication that the computer was still running the program. Address 22CH is the beginning of the ROM routine to blink the asterisk in the upper right-hand corner of the screen. As an afterthought, I could have added an LED to the EPROM circuit as an indicator that programming pulses were in progress, but why add extra expense?

Final Thoughts

You may want to add pull-up resistors to all of the address and data lines, depending upon how noisy your computer's environment is. In Photo 1 you can see that I did include them (labeled 4.7k), but I've since found out that the board works just as well without them, so they were omitted from the schematic (Figure 5). If you include them, I would highly recommend resistor networks as shown, because wire-wrapping 19 individual resistors would be an unnecessary headache. If multiple EPROMs are to be programmed at one sitting, a normally-open push-button switch could be added in place of C1, so that U5 could be reset manually at the beginning of each run.

Hopefully you can now build the programmer presented here or design your own from the information presented. Once you have a working unit, the possibilities for programs to put into the EPROM and have on line at all times are limited only by the 2000-byte size of the free memory space.

Program Listing. *Sample program listing in Editor/Assembler format.*

```
00100 SETUP  ORG    8000H      ;YOU CAN START ANYWHERE
00110        LD     SP,8000H   ;SET UP YOUR OWN STACK (OPTIONAL)
00120        LD     HL,9000H   ;ADDRESS OF PROGRAM TO BE LOADED
00130        LD     DE,3000H   ;FIRST ADDRESS OF EPROM
00140        LD     BC,XXXXX   ;XXXXX IS THE NUMBER OF BYTES
00150                          ;IN YOUR PROGRAM(S)
00160 START  PUSH   BC         ;SAVE BYTE COUNT ON STACK
00170        LD     A,(HL)     ;GET BYTE TO BE PROGRAMMED
00180        LD     (DE),A     ;SEND IT TO EPROM BOARD (LATCHES)
00190        IN     A,(0)      ;TOGGLE PROGRAMMING PULSE ON
00200        LD     BC,3500    ;LOAD A DELAY COUNT INTO BC
00210        CALL   60H        ;CALL ROM DELAY LOOP - COUNT OF
00220                          ;3500 WILL DELAY ABOUT 51.3 MSEC
00230        IN     A,(0)      ;TOGGLE PROGRAMMING PULSE OFF
00240        CALL   22CH       ;ROM ROUTINE TO BLINK ASTERISK
00250        INC    HL         ;POINT TO NEXT BYTE TO BE PROGRAMMED
00260        INC    DE         ;POINT TO NEXT ADDRESS IN EPROM
00270        POP    BC         ;GET THE BYTE COUNT BACK
00280        DEC    BC         ;ONE LESS BYTE TO GO
00290        LD     A,B        ;THESE THREE STEPS CHECK TO SEE
00300        OR     C          ;  IF THE BC REGISTER IS ZERO (DONE)
00310        JR     NZ,START   ;  AND WILL JUMP BACK TO START IF NOT
00320 DONE   JP     YYYYY      ;OTHERWISE WE'RE DONE AND CAN JUMP
00330                          ;OUT OF THIS ROUTINE - E.G., 1A19H = 'READY',
00340                          ;402DH = 'DOS READY', ETC.
00350        END
```

HOME APPLICATIONS

Pari-Mutuel
Income Tax Withholding

Pari-mutuel

by Arthur Welcher

No, this isn't another program for making animated horses race around your CRT. It is not a program with random numbers to bet on. It is a program to bet with!

Some friends, all of us racing fans, gathered to watch the Kentucky Derby telecast. We decided we would like to bet on the race, just between ourselves, of course. Normally, the betting would be done as a pool, where everyone puts in a dollar and draws a number out of a hat to get the number of their horse. But we wanted to include win, place, and show bets, and the ability to bet on more than one horse in each race in our betting. We were not restricted to the pool system—there were enough of us to set up a pari-mutuel betting system. What does pari-mutuel mean? I'm glad you asked, because that is what this program is all about.

In the good old days, you had to wager with a bookmaker. The bookmaker would handicap, or rate, the horses in the race himself and set his own odds on each horse's chance of winning. The more money bet with him on one horse, the lower each new bettor's odds would be—the bookmaker didn't want to pay out more money on a race than he collected. He may have given one person odds of 6 to 1 on a horse, and ten minutes later given someone else only 3 to 1 on that same horse. More money had been bet on that horse, proportionately, in those ten minutes than on the other horses. The odds got higher as the bookmaker got fewer bets on a particular horse.

The obvious solution was to have a controlled system where all bettors get the same payoff, regardless of whether they bet early or late. That system, known as the pari-mutuel system of wagering, places all of the money bet on each horse in a race into a pool. The odds are calculated on each horse, periodically, in the following manner.

The total pool is reduced by about 18 percent, depending on the state in which the race is held. The remainder becomes the payout pool. From that figure, subtract the amount bet on one horse then divide what is left by the amount bet on that horse. You now have the odds on that particular horse. The process must be repeated for each horse in the race. When a race is started, the machines that print the tickets are automatically locked out, and the final calculations are made. The win, place, and show pools are each figured separately. The payout place pool must be divided by two, half of the pool going to pay the place price on the winner, the other half to pay the price for the place horse, or the horse that finishes second. The payout show pool is handled in the same way, except it is divided by three, for the first three horses that finish. The race track uses the 18 percent withheld for operating expenses, purses, etc., with a share going to the state for taxes.

When all the mathematics have been done, the actual payoff figure is rounded down to the nearest 10 cents. That means that as much as nine cents of each payoff per dollar will not be divided among the winning tickets. This becomes the "breakage," a sizeable sum after a few hundred thousand dollars have been wagered. This money becomes a part of the track's income.

Since these formulas calculate the payoff per dollar, and the minimum bet is two dollars, we must build a "times 2" factor into our formula so the posted payoff amount is for a two dollar ticket.

In the event that there is a "minus" pool (so much money bet on the winning horse that the payout pool is not large enough to pay 10 cents per dollar), the race track must dip into its funds to make up the difference to make the payoff 10 cents on the dollar or a $2.20 payoff. In some cases, even a $2.10 payoff is the minimum.

For our purposes, many of the historic elements discussed were not necessary. We didn't need to take out a cut for the state or for operating expenses. We wanted to divide and pay out the entire pool; therefore there was no breakage. We did, however, follow the track's example of placing a unique code on all tickets for each race, so that no one can use a losing ticket on horse number 3 in the first race yesterday as a winning ticket on horse number 3 in the first race today.

By the Preakness party, we had a program working that gave us a complete win, place, and show pari-mutuel betting system. (See Program Listing.) We placed a roll of adding machine paper in our printer to make the tickets. We used a TV modulator so that we could flash odds and pool totals on the TV screen periodically as the tickets were sold. This information would also appear on the computer CRT. One person acted as ticket seller at the TRS-80 keyboard. If you have some sort of horse race game, such as the dice-throwing type, this program works beautifully!

Line 110 lets you set options for the maximum number of horses in any one race. For the dice game mentioned here this would be six. Most racetracks usually limit the number to 12.

Line 120 lets you select the number of wagering pools you will use. To simulate a racetrack using win, place, and show pools, select three. For the dice game, select one for win only.

Line 130 sets the dimensions for the arrays used in the program, the first element being the number of horses, the second being the number of pools. Lines 150 and 155 set all pool totals to zero so that you may start the wagering. Line 200 is the random code generator for each race.

Lines 1000 to 1270 sell a ticket, print it out, and add the amount to the proper wagering pool. The computer asks for HORSE NO. You input the requested number. The computer then asks WIN, PLACE OR SHOW. Enter either a 1, 2, or 3. (Note: If you have opted for a win only race, this step does not appear.) The computer then asks for AMOUNT. Respond by inputting the dollar amount of your bet. The computer prints the ticket and records the wager.

You may continue to print tickets or select POST ODDS which will be updated with each win ticket sale. The place and show pool information is visible for anyone who wants to see it, but is not reflected in the odds. This is also true at the racetracks.

The odds are calculated in line 2020. There are two math possibilities that must be discussed here. If there is no money bet on one horse, dividing by zero would cause an error. (It would make the odds infinity to one!) Line 2015 sets odds in this event to be 99 to 1 (a standard racetrack maximum). If the pool is approaching a minus pool, line 2025 sets our lowest odds at 1 to 1 (definitely not a race track minimum). The track minimum is 1 to 9, or 11 cents to the dollar, but with breakage becomes 10 cents, or a $2.20 minimum payoff.

After the race, lines 3010, 3020, and 3030 ask for the results in order of finish. When you input these numbers, lines 3055 to 3130 calculate the payoff. Line 3150 sets the screen for the expanded mode, and lines 3160 to 3220 print the prices on the screen.

Remember, pari-mutuel betting works best for fairly large-scale betting. With many people betting and money being bet on each horse in each pool, there is usually not a great percentage difference between the largest amount bet on one horse and the smallest amount bet on another horse in that pool. Odds usually vary between 2 to 1 and 50 to 1. If you have unusually large differences, you will see unusual odds, but that's what makes horse racing!

Program Listing. *Pari-mutuel*

Encyclopedia Loader

```
 50 :
    ' *******************************************
 55 :
    ' *                                         *
 60 :
    ' *            PARIMUTUEL                    *
 65 :
    ' *              BY                         *
 70 :
    ' *           ARTHUR WELCHER                *
 75 :
    ' *                                         *
 80 :
    ' *******************************************
 90 CLS
100 CLEAR 500
110 PRINT @400,"";:
    PRINT "ENTER NO. OF HORSES USED ";:
    INPUT H
120 PRINT @464,"";:
    PRINT "ENTER  1. IF WIN ONLY
                  2. IF WIN & PLACE
                  3. IF WIN PLACE AND SHOW":
    GOSUB 5000:
    B = KB
130 DIM A(H,B):
    DIM AM(H,B)
140 REM
150 FOR P = 1 TO H:
     FOR Q = 1 TO B:
      A(P,Q) = 0:
      NEXT Q,P
155  TW = 0:
     TP = 0:
     TS = 0
160  R = R + 1
200  CLS :
     X0 = RND(95) + 32:
     X1 = RND( 95) + 32:
     X2 = RND( 95) + 32:
     X3 = RND( 95) + 32:
     X4 = RND( 95) + 32:
     CO$ = CHR$(X0) + CHR$(32) + CHR$(X1) + CHR$(32) + CHR$(X2)
     + CHR$(32) + CHR$(X3) + CHR$(32) + CHR$(X4)
210  CLS :
     PRINT @400,"  M  E  N  U"
220  PRINT :
     PRINT TAB(12)"1. SELL TICKET"
230  PRINT TAB(12)"2. POST ODDS"
240  PRINT TAB(12)"3. POST PAYOFF"
250  PRINT :
     PRINT TAB(12)"SELECTION :"
260  GOSUB 5000
270  ON KB GOTO 1000,2000,3000
275  GOTO 210
280  REM
1000 :
     ' *********SELL TICKET *******
1010 HN$ = "HORSE NO.":
     UH$ = "$###.##"
1020 CLS :
     PRINT @400,"SELL TICKETS FOR RACE NO. ";R:
     PRINT
1030 PRINT @528,HN$:
     GOSUB 5000:
     H1 = KB:
     IF B = 1
```

```
         THEN
          W = 1:
          GOTO 1100
1040    IF B = 3
         THEN
          1070
1050    PRINT @528,HN$;H1;" WIN OR PLACE ";:
        GOSUB 5000:
        W = KB
1060    GOTO 1100
1070    PRINT @528,HN$;H1;" WIN, PLACE OR SHOW ";:
        GOSUB 5000:
        W = KB
1100    IF W = 1
         THEN
          W$ = " TO WIN "
1110    IF W = 2
         THEN
          W$ = " TO PLACE "
1120    IF W = 3
         THEN
          W$ = " TO SHOW "
1125    PRINT @528,HN$;H1;W$;"                          ";
1130    PRINT @528,HN$;H1;W$;"      AMOUNT ";:
        INPUT AM(H1,W)
1135    REM IF W>H THEN 1030
1140    A(H1,W) = A(H1,W) + AM(H1,W):
        TW = TW + AM(H1,1):
        TP = TP + AM(H1,2):
        TS = TS + AM(H1,3)
1150    LPRINT CHR$(138):
        LPRINT CHR$(27); CHR$(14)"  WELCHER"
1160    LPRINT CHR$(27); CHR$(14)"   PARK"
1170    LPRINT CHR$(138):
        LPRINT STRING$(20,"=")
1180    LPRINT "  RACE NO. ";R
1190    LPRINT CHR$(138):
        LPRINT CHR$(27); CHR$(14);CO$
1200    LPRINT "":
        LPRINT STRING$(20,"=")
1210    LPRINT CHR$(138):
        LPRINT "   AMOUNT ";:
        LPRINT USING UH$;AM(H1,W)  :
        LPRINT CHR$(138):
        LPRINT "  " ; CHR$(27); CHR$(14)W$
1220    LPRINT CHR$(138):
        LPRINT STRING$(20,"-")
1225    AM(H1,1) = 0:
        AM(H1,2) = 0:
        AM(H1,3) = 0
1230    PRINT @912,"1. ANOTHER"
1240    PRINT TAB(16)"2. RETURN TO MENU"
1250    PRINT TAB(8)"SELECTION :";
1260    GOSUB 5000
1270    ON KB GOTO 1000,210
1275    GOTO 1020
2000    :
        ' ******* POST ODDS ********
2010    FOR N = 1 TO H
2015     IF A(N,1) = 0
          THEN
           OD(N) = 99:
           GOTO 2030
2020     OD(N) = INT((TW - A(N,1)) / A(N,1))
2025     IF OD(N) < 1
          THEN
           OD(N) = 1
2030     NEXT
2040    CLS :
        PRINT HN$ TAB(10)"ODDS","  WIN POOL","PLACE POOL","SHOW POOL"
2050    PRINT STRING$(64,"=")
```

Program continued

```
2060  FOR N1 = 1 TO H
2070   PRINT TAB(3)N1; TAB(10)OD(N1),"      ";A(N1,1),"    ";A(N1,2),"
       ";A(N1,3)
2080   NEXT N1
2090  PRINT :
      PRINT "TOTAL","      ";TW, "   ";TP,"   ";TS:
      PRINT :
      PRINT
2100  PRINT TAB(16)"1. SELL TICKET"
2110  PRINT TAB(16)"2. POST PAYOFF"
2120  GOSUB 5000
2130  ON KB GOTO 1000,3000
2135  GOTO 2040
3000  :
      ' ******* POST PAYOFF ********
3010  CLS :
      PRINT @400,"";:
      INPUT " WINNER ";E
3020  PRINT TAB(16)"";:
      INPUT " SECOND ";F
3030  PRINT TAB(16)"";:
      INPUT " THIRD ";G
3040  FOR X9 = 1 TO 200:
       NEXT
3050  CLS
3055  IF A(E,1) = 0
       THEN
        E1 = TW:
        GOTO 3070
3060  E1 = INT((TW - A(E,1)) / A(E,1)) * 2
3070  P2 = (TP - (A(E,2) + A(F,2)) / 2)
3075  IF A(E,2) = 0
       THEN
        E2 = P2:
        GOTO 3085
3080  E2 = INT(P2 / A(E,2)) * 2
3085  IF A(F,2) = 0
       THEN
        F2 = P2 :
        GOTO 3100
3090  F2 = INT(P2 / A(F,2)) * 2
3100  P3 = (TP - (A(E,3) + A(F,3) + A(G,3)) / 3)
3105  IF A(E,3) = 0
       THEN
        E3 = P3:
        GOTO 3115
3110  E3 = INT(P3 / A(E,3)) * 2
3115  IF A(F,3) = 0
       THEN
        F3 = P3:
        GOTO 3125
3120  F3 = INT(P3 / A(F,3)) * 2
3125  IF A(G,3) = 0
       THEN
        G3 = P3:
        GOTO 3150
3130  G3 = INT(P3 / A(G,3)) * 2
3150  PRINT CHR$(23)
3155  IF B = 1
       THEN
        E2 = 0:
        E3 = 0:
        F2 = 0:
        F3 = 0:
        G3 = 0
3156  IF B = 2
       THEN
        E3 = 0:
        F3 = 0:
        G3 = 0
3160  PRINT "HORSE" TAB(8)"WIN" TAB(16)"PLACE" TAB(25)"SHOW":
      PRINT
3170  UE$ = " #     $###.##  $###.##  $###.##"
3180  PRINT USING UE$;E,E1,E2,E3
3190  UF$ = " #              $###.##  $###.##"
3200  PRINT USING UF$;F,F2,F3
3210  UG$ = " #                       $###.##"
3220  PRINT USING UG$;G,G3
3230  PRINT :
      PRINT :
      PRINT
3240  INPUT "FOR NEXT RACE HIT ENTER";Q:
      CLEAR :
      CLS :
      R = R + 1:
      GOTO 150
5000  KB$ = INKEY$:
      IF KB$ = ""
       THEN
        5000
5010  KB = VAL(KB$):
      RETURN
```

Income Tax Withholding

by Cliff DeJong

A few years back, my wife went into real estate and was quite successful. I didn't realize how successful until our federal income taxes were due. We owed an additional payment of nearly $2,000! I had mistakenly assumed that we would have sufficient withholding from my salary. After all, we had five children and above-average deductions.

The following year, I reduced my withholding allowances to increase the federal tax withheld. Real estate fell off a little, however, and we ended up getting a refund of about $1,000. Neither extreme of $2,000 due or $1,000 refund was desirable. I don't like borrowing from or lending money to the federal government. In talking to several friends, I found that a common problem for families or individuals with two or more sources of income is making sure that the federal income tax withheld is sufficient to cover your tax bill on April 15. One way to guess at a solution is to obtain a federal form W-4 from payroll. Form W-4 is the Employee's Withholding Allowance Certificate. It contains instructions to allow you to estimate the number of withholding allowances you should claim. Each withholding allowance, or exemption claimed, is equivalent to either an exemption or, in 1980, an extra $1,000 in deductions.

The W-4 instructions, however, are confusing and at best only approximate. In addition, the W-4 form does not give estimates of the refund resulting from deliberate overwithholding. There is no guidance for a change in withholding allowances during the year, either. This program (see Program Listing) has solved my income tax withholding problems and has also helped several of my friends. It is written for a 16K Level II TRS-80 and could easily be adapted to other computers.

Overview

This tax withholding program collects data on projected income from several sources and combines it with estimates of itemized deductions, tax credits, tax losses, and the number of exemptions. From this data, the estimated federal income tax liability is computed. A table is then displayed for each wage earner showing the refund or balance due for different withholding allowances. Also shown is the change in weekly take-home pay. The program accounts for taxes withheld to date and allows for changes in income or withholding allowances, etc. during the year. Filing status may be Single, Married Filing Jointly, or Married Filing Separately.

Estimated tax liability is based on federal form 1040-ES, Declaration of Estimated Tax for Individuals. Computation of withholding rates is done by the federal percentage method. All tables are based on rates for the 1980 tax year. As of this writing, these tables are correct, but may be revised. In any event, the correct tables are available through your local IRS office. The program provides an option for output to a printer and allows data to be stored on cassette tape. The tape data can be read back in and altered, as necessary, later in the year.

Method

The code begins by displaying a title and initializing the standard deductions and the value of an exemption. The date and filing status are input, and the number of weeks remaining in the year is computed. Income data is then input from the keyboard or from tape. For these inputs, it is helpful to have your most recent pay stub. For each wage earner, the user inputs the tax withheld to date, the current number of withholding allowances claimed, and the current rate of pay. The withholding income is computed as the current annual rate of pay minus the withholding allowances multiplied by $1,000. The annual withholding rate is computed using Table 1 or Table 2, as appropriate, based on the withholding income.

If the income data is input from tape, the data for each income source is displayed and you are asked if the data is correct. If not, you may enter the

Withholding Income		Amount of Income Tax to be Withheld	
Over—$2400	But not over—$6600	15%	Of the excess over—$2400
$6600	$10900	$630+18%	$6600
$10900	$15000	$1404+21%	$10900
$15000	$19200	$2265+24%	$15000
$19200	$23600	$3273+28%	$19200
$23600	$28900	$4505+32%	$23600
$28900	—	$6201+37%	$28900

Table 1. *Withholding rate: married person*

Withholding Income		Amount of Income Tax to be Withheld	
Over—$1420	But not over—$3300	15%	Of the excess over—$1420
$3300	$6800	$282+18%	$3300
$6800	$10200	$912+21%	$6800
$10200	$14200	$1626+26%	$10200
$14200	$17200	$2666+30%	$14200
$17200	$22500	$3566+34%	$17200
$22500	—	$5368+39%	$22500

Table 2. *Withholding rate: single person*

correct data from the keyboard. A summary of all income data is then printed for verification. The program next asks for additional tax information, such as an estimate of itemized deductions and the number of exemptions. I use last year's deductions as a basis for my estimates and guess a little higher or lower. As before, if the data is from tape, it is displayed and corrected if necessary. Tax liability is then computed according to either Table 3 or Table 4, which give the tax rate schedules for Married Filing Jointly and Single, respectively. For a filing status of Married Filing Separately, the Married Filing Jointly table is used with the taxable income and the tax, both divided by two.

Taxable Income		Tax	
Over—$3400	But not over—$5500	14%	Of the amount over—$3400
$5500	$7600	$294+16%	$5500
$7600	$11900	$630+18%	$7600
$11900	$16000	$1404+21%	$11900
$16000	$20200	$2265+24%	$16000
$20200	$24600	$3273+28%	$20200
$24600	$29900	$4505+32%	$24600
$29900	$35200	$6201+37%	$29900
$35200	$45800	$8162+43%	$35200
$45800	$60000	$12720+49%	$45800
$60000	$85600	$19678+54%	$60000
$85600	$109400	$33502+59%	$85600
$109400	$162400	$47544+64%	$109400
$162400	$215400	$81464+68%	$162400
$215400	—	$117504 + 70%	$215400

Table 3. *1980 Tax rate schedules: married filing jointly*

Taxable Income		Tax	
Over—$2300	But not over—$3400	14%	Of the amount over—$2300
$3400	$4400	$154+16%	$3400
$4400	$6500	$314+18%	$4400
$6500	$8500	$692+19%	$6500
$8500	$10800	$1072+21%	$8500
$10800	$12900	$1555+24%	$10800
$12900	$15000	$2059+26%	$12900
$15000	$18200	$2605+30%	$15000
$18200	$23500	$3565+34%	$18200
$23500	$28800	$5367+39%	$23500
$28800	$34100	$7434+44%	$28800
$34100	$41500	$9766+49%	$34100
$41500	$55300	$13392+55%	$41500
$55300	$81800	$20892+63%	$55300
$81800	$108300	$37677+68%	$81800
$108300	—	$55697+70%	$108300

Table 4. *1980 Tax rate schedules: single*

The taxable income is the gross income less, the number of exemptions multiplied by $1,000 (in 1980), less excess deductions. Excess deductions are those amounts above the standard deduction. Tax losses are also subtracted. The calculated income tax liability is then reduced by tax credits to yield net tax liability. The total amount of withholding from all sources can then be calculated easily, and using the tax liability figure calculated earlier, the refund or balance due at the end of the year can be estimated.

The program also computes the refund or balance due if the number of withholding allowances, that is, exemptions claimed, is changed from the input date to the end of the year. This calculation is made for each income source subject to withholding and is done for withholding allowances ranging from zero to 10. As a bonus, the net change in weekly take-home pay is also shown for different numbers of withholding allowances.

A$	temporary string variable
*AE	number of exemptions
AW	annual withholding
CW	current withholding rate, $/week
D	current day
DD	day
DE	excess deductions
*DI	itemized deductions
DM	dimension for income sources
*DT	day of tape data
E	exemptions claimed index
*EC(I)	current exemptions claimed
*EW(I)	estimated annual withholding
EX	value of an exemption ($1000 in 1980)
F	temporary variable
F1	temporary variable
FS	filing status flag (0 is MJ, 1 is S, 2 is MS)
GI	total gross income
GW	total annual withholding
I	index variable
*ID$(I)	identifier for income source
IT	flag for input (1 is tape, 0 is keyboard)
J	index variable
*LT	tax losses
M	current month
MM	month
*MT	month of tape data
*NS	number of income sources
P	flag for printer output (0 is no, 1 is yes)
R	estimated refund
*RW	income rate, $/week
SD(FS)	standard deduction for filing status FS
TA	temporary variable
TB	tax bracket
TC	tax credits

Table continued

TE	temporary variable
TI	taxable income
TL	temporary variable
*TW(I)	tax withheld to date
TX	tax liability
WA	temporary variable
WB	temporary variable
WH	withholding without income source I
WI	withholding income
WL	temporary variable
WR	weeks remaining to end of year
WU	weeks since update
WW	current weekly withholding
Y	current year
*YT	year of tape data
YY	year

Table 5. *Variable list*

Sample Problem

The use of the tax withholding program can best be illustrated by a sample problem. Table 6 shows the dialogue between the program and Sugar Daddy. Sugar Daddy has his and his wife's pay stubs for April 2, and an estimate of their deductions for 1980. His pay stub shows $6,000 gross income to date and a gross income rate of $650 per week. $1,500 has been withheld for federal taxes, and Sugar Daddy is currently claiming four exemptions. His wife, Sweetie Pie, has earned $2,500 so far and is drawing $225 per week. She has had $600 withheld and is claiming no exemptions. Additional income expected is $300 from bank interest on a savings account.

The income summary computed by the program shows that Sugar Daddy and Sweetie Pie are doing quite well, with $42,925 gross income expected in 1980. Estimated withholding is $8,180 for the current withholding allowances. Itemized deductions are estimated by Sugar Daddy to be $6,000 for 1980, with a tax credit of $125 and tax losses of $300. Adding in their two children, Sugar Daddy and Sweetie Pie are entitled to four federal income tax exemptions.

With the above information, the program estimates the total tax liability as $8,391. The table labeled FOR INCOME FROM SUGAR DADDY shows that Sugar Daddy will have an additional payment of $212 due to the IRS if he continues to claim four exemptions. However, if Sugar Daddy claims only one exemption from April 2 on, he will see a refund of $621 from IRS. In that case, Sugar Daddy's take-home pay will decrease by $21 each week. Other options are evident from the printout. Since Sweetie Pie is already claiming no exemptions, any change in her withholding status would simply increase the balance due.

Modifications and Updates

The program as written is correct for the 1979 and 1980 tax years, but may need to be changed. The current tables will be available from your local IRS office or, possibly, you can borrow them from your company payroll office. The information in Tables 1, 2, 3, and 4 is in the program at data statements numbered 1740, 1780, 1830, and 1870, respectively. The maximum income is represented in the second column, and the corresponding percentage in the third column. In addition, the standard deductions and value of an exemption are in line 1690.

```
OUTPUT TO PRINTER? NO
DATE (MM,DD,YYYY)? 4,2,1980
FILING STATUS?
    MJ = MARRIED FILING JOINTLY
    MS = MARRIED FILING SEPARATELY
    S = SINGLE
? MJ

INCOME DATA
INPUT DATA FROM KEYBOARD(K) OR TAPE(T)? K

INPUT IDENTIFIER FOR INCOME (E.G., 'JOAN')
    PRESS ENTER IF NO MORE SOURCES? SUGAR DADDY

FOR INCOME FROM SUGAR DADDY
GROSS INCOME TO DATE? 6000
ESTIMATED INCOME FOR REMAINDER OF YEAR ($/WK)? 650
TAX WITHHELD TO DATE? 1500
CURRENT EXEMPTIONS CLAIMED? 4

INPUT IDENTIFIER FOR INCOME (E.G.,'JOAN' )
    PRESS ENTER IF NO MORE SOURCES? SWEETIE PIE

FOR INCOME FROM SWEETIE PIE
GROSS INCOME TO DATE? 2500
ESTIMATED INCOME FOR REMAINDER OF YEAR ($/WK)? 225
TAX WITHHELD TO DATE? 600
CURRENT EXEMPTIONS CLAIMED? 0
INPUT IDENTIFIER FOR INCOME (E.G.,'JOAN' )
    PRESS ENTER IF NO MORE SOURCES? BANK INTEREST

FOR INCOME FROM BANK INTEREST
GROSS INCOME TO DATE? 300
ESTIMATED INCOME FOR REMAINDER OF YEAR ($/WK)? 0
TAX WITHHELD TO DATE? 0

INPUT IDENTIFIER FOR INCOME (E.G., 'JOAN')
    PRESS ENTER IF NO MORE SOURCES? <ENTER>
```

Table continued

```
SUMMARY OF INCOME
                     ESTIMATED          CURRENT            ESTIMATED
SOURCE             ANNUAL INCOME  EXEMPTIONS CLAIMED     WITHHOLDING
---------------------------------------------------------------------
SUGAR DADDY          $ 31350              4               $ 6401
SWEETIE PIE          $ 11275              0               $ 1779
BANK INTEREST        $   300              0               $    0
                     -------                              ------
TOTALS               $ 42925                              $ 8180

PRESS ENTER TO CONTINUE? <ENTER>

ADDITIONAL TAX INFORMATION
ITEMIZED DEDUCTIONS ( 0 FOR STANDARD DEDUCTION)? 6000
TAX CREDITS? 125
TAX LOSSES (INPUT AS + )? 300
TOTAL NUMBER OF EXEMPTIONS? 4

TAX LIABILITY = $ 8391                                   43 % BRACKET

   FOR INCOME FROM SUGAR DADDY

   EXEMPTIONS    ESTIMATED REFUND ( + )    CHANGE IN TAKE-HOME
    CLAIMED       OR BALANCE DUE (-)          PAY PER WEEK
      0                $  899                   $ 28-
      1                $  621                   $ 21-
      2                $  344                   $ 14-
      3                $   66                   $  7-
      4                $  212-                  $  0
      5                $  485-                  $  7
      6                $  725-                  $ 13
      7                $  965-                  $ 19
      8                $ 1205-                  $ 25
      9                $ 1445-                  $ 32
     10                $ 1685-                  $ 38
PRESS ENTER TO CONTINUE?

TAX LIABILITY = $ 8391
   FOR INCOME FROM SWEETIE PIE                           43 % BRACKET

   EXEMPTIONS    ESTIMATED REFUND ( + )    CHANGE IN TAKE-HOME
    CLAIMED       OR BALANCE DUE (-)          PAY PER WEEK
      0                $  212-                  $  0
      1                $  365-                  $  4
      2                $  500-                  $  7
      3                $  635-                  $ 11
      4                $  770-                  $ 14
      5                $  905-                  $ 18
      6                $ 1019-                  $ 21
      7                $ 1132-                  $ 24
      8                $ 1244-                  $ 26
      9                $ 1357-                  $ 29
```

```
        10                 $ 1391-                    $ 30
PRESS ENTER TO CONTINUE?

SAVE DATA TO TAPE? NO
READY
>
```

Table 6. *Sample program*

If you are fortunate enough to have a disk drive, the program can be easily modified to save the data to disk rather than to cassette. The necessary changes are shown in Table 7. These consist of adding OPEN and CLOSE statements, adding the inputs for the filespec, changing INPUT#-1 to INPUT#1 and PRINT#-1 to PRINT#1, and adding a comma to the disk output file to separate the output string from the rest of the data.

An enhancement that would be useful is adding state income tax considerations. I did not put that into the program because it is a relatively small effect, compared to federal income tax, and there are 50 states! Moreover, most state income taxes are based strongly on the federal tax, so adjusting exemptions for federal taxes will work for state taxes also.

Change the following statements as shown:

```
360   INPUT"INPUT DATA FROM KEYBOARD(K) OR DISK (D)";A$:
      A$=LEFT$(A$,1)
370   IF A$="K" THEN 580 ELSE IF A$<>"D" THEN 360
390   'INPUT FROM DISK
410   IT=1:PRINT:INPUT"FILESPEC";FS$:OPEN "I",1,FS$
420   INPUT#1,NS,DI,TC,LT,AE,MT,DT,YT:IF NS>DM THEN 650
450   INPUT#1,W(I),ID$(I);RW(I),EC(I),TW(I)
480   CLS:PRINT"DISK DATE : ";MT;"/";DT;"/";YT:PRINT
560   NEXT I:CLOSE 1:GOTO 670
1340  'SAVE DATA TO DISK
1360  CLS:INPUT "SAVE DATA TO DISK";A$
1380  PRINT:INPUT"FILESPEC";FS$:OPEN "O",1,FS$
1400  PRINT#1,NS,DI,TC,LT,AE,M,D,Y
1420  PRINT#1,W(I),ID$(I),",",RW(I),EC(I),TW(I)
1430  NEXT I:CLOSE 1
```

Table 7. *Disk modifications*

Program Listing. *Income tax withholding*

Encyclopedia Loader

```
10 :
   '  TAX WITHHOLDING PROGRAM
20 :
   '  BY CLIFF DE JONG  COLORADO SPRINGS, COLORADO
30 :
   '
40 POKE 16553,255:
   CLEAR 1000
50 :
   '
60 :
   '  DM IS THE MAXIMUM NUMBER OF INCOME SOURCES
70 :
   '
80 DM = 10:
   DIM W(DM),ID$(DM),RW(DM),EC(DM),EW(DM),TW(DM),SD(2)
90 CLS :
   PRINT @405,"TAX WITHHOLDING PROGRAM":
   PRINT @543,"BY":
   PRINT @666,"CLIFF DE JONG"
100 FOR I = 1 TO 2000:
     NEXT :
    GOSUB 1670:
    F$ = "######"
110 CLS :
    INPUT "OUTPUT TO PRINTER";A$:
    PRINT :
    IF LEFT$(A$,1) = "Y"
     THEN
      P = 1:
     ELSE
      130
120 IF PEEK(14312) > 127
     THEN
      INPUT "TURN ON PRINTER, PRESS ENTER";A$:
      GOTO 120
130 PRINT :
    INPUT "DATE(MM,DD,YYYY)";M,D,Y
140 PRINT :
    PRINT "FILING STATUS?":
    PRINT "  MJ = MARRIED FILING JOINLY":
    PRINT "  MS = MARRIED FILING SEPARATELY":
    PRINT "  S = SINGLE":
    INPUT A$
150 IF A$ = "MJ"
     THEN
      FS = 0:
      GOTO 200
160 IF A$ = "MS"
     THEN
      FS = 2:
      GOTO 200
170 IF A$ = "S"
     THEN
      FS = 1:
      GOTO 200
180 GOTO 140
190 :
    '
200 :
    ' COMPUTE NUMBER OF WEEKS REMAINING (WR)
210 :
    '
220 MM = M:
    DD = D:
    YY = Y:
    GOSUB 1590:
```

```
    F1 = F
230 MM = 12:
    DD = 31:
    YY = Y:
    GOSUB 1590:
    WR = (F - F1) / 7
240 :
    '
250 :
    ' PRINT OUT DATE AND FILING STATUS
260 :
    '
270 IF P = 0
     THEN
      330
280 LPRINT :
    LPRINT M;"/";D;"/";Y:
    LPRINT
290 IF FS = 0
     THEN
      LPRINT "MARRIED FILING JOINTLY":
      LPRINT :
      GOTO 330
300 IF FS = 1
     THEN
      LPRINT "SINGLE":
      LPRINT :
      GOTO 330
310 IF FS = 2
     THEN
      LPRINT "MARRIED FILING SEPARATELY":
      LPRINT
320 :
    '
330 :
    ' INPUT INCOME DATA
340 :
    '
350 CLS :
    PRINT "INCOME DATA":
    PRINT
360 INPUT "INPUT DATA FROM KEYBOARD(K) OR TAPE(T)";A$:
    A$ = LEFT$(A$,1)
370 IF A$ = "K"
     THEN
      580:
     ELSE
      IF A$ < > "T"
       THEN
        360
380 :
    '
390 :
    ' INPUT FROM TAPE
400 :
    '
410 IT = 1:
    PRINT :
    PRINT "PREPARE CASSETTE, PRESS ENTER":
    INPUT A$
420 INPUT # - 1,NS,DI,TC,LT,AE,MT,DT,YT:
    IF NS > DM
     THEN
      650
430 MM = MT:
    DD = DT:
    YY = YT:
    GOSUB 1590:
    WU = (F1 - F) / 7
440 FOR I = 1 TO NS
```

Program continued

```
450  INPUT # - 1,W(I),ID$(I),RW(I),EC(I),TW(I)
460  POKE 16553,255
470  WI = 52 * RW(I) - EC(I) * EX:
     GOSUB 1970:
     TW(I) = TW(I) + WW * WU:
     EW(I) = TW(I) + WW * WR
480  CLS :
     PRINT "TAPE DATE : ";MT;"/";DT;"/";YT:
     PRINT
490  PRINT "FOR INCOME FROM ";ID$(I):
     PRINT
500  PRINT "  ESTIMATED ANNUAL INCOME = $";W(I)
510  PRINT "  CURRENT INCOME RATE = ";RW(I);" $ PER WEEK"
520  IF TW(I) > 0
      THEN
       PRINT "  ";EC(I);" EXEMPTIONS CURRENTLY CLAIMED"
530  PRINT "  TAXES WITHHELD TO DATE = $";TW(I)
540  PRINT :
     INPUT "IT THIS CORRECT";A$
550  IF LEFT$(A$,1) = "N"
      THEN
       GOSUB 1470
560  NEXT I:
    GOTO 670
570 :
    '
580 :
    ' INPUT FROM KEYBOARD
590 :
    '
600 IT = 0:
    I = 1
610 CLS :
    PRINT "INPUT IDENTIFIER FOR INCOME (E.G.,'JOAN')"
620 ID$(I) = "":
    INPUT "  PRESS ENTER IF NO MORE SOURCES";ID$(I):
    IF ID$(I) = ""
     THEN
      NS = I - 1:
      GOTO 670
630 GOSUB 1470
640 I = I + 1:
    IF I < DM
     THEN
      610
650 PRINT :
    PRINT "ONLY ";DM;" INCOME SOURCES ALLOWED":
    PRINT "INCREASE DM IN LINE 60 TO MAXIMUM NUMBER OF INCOME SOURCE
    S AND RE-RUN":
    STOP
660 :
    '
670 :
    ' PRINT SUMMARY OF INCOME
680 :
    '
690 CLS :
    PRINT "SUMMARY OF INCOME"
700 PRINT TAB(19)"ESTIMATED"; TAB(37)"CURRENT"; TAB(53)"ESTIMATED"
710 PRINT "SOURCE"; TAB(17)"ANNUAL INCOME"; TAB(32)"EXEMPTIONS CLAIM
    ED"; TAB(52)"WITHHOLDING":
    PRINT STRING$(63,"-")
720 GI = 0:
    GW = 0:
    FOR I = 1 TO NS
730  PRINT ID$(I); TAB(20)"$";:
     PRINT USING F$;W(I);:
     PRINT TAB(40)EC(I); TAB(54)"$";:
     PRINT USING F$;EW(I)
740  GI = GI + W(I):
      GW = GW + EW(I):
      NEXT I
 750 PRINT TAB(21)"------"; TAB(55)"------"
 760 PRINT "TOTALS"; TAB(20)"$";:
     PRINT USING F$;GI;:
     PRINT TAB(54)"$";:
     PRINT USING F$;GW
 770 IF P = Ø
      THEN
       85Ø
 780 LPRINT :
     LPRINT "SUMMARY OF INCOME :":
     LPRINT STRING$(17,"-"):
     LPRINT :
     LPRINT TAB(19)"ESTIMATED"; TAB(37)"CURRENT"; TAB(53)"ESTIMATED"
 790 LPRINT "SOURCE"; TAB(17)"ANNUAL INCOME"; TAB(32)"EXEMPTIONS CLAI
     MED"; TAB(52)"WITHHOLDING":
     LPRINT STRING$(63,"-")
 800 FOR I = 1 TO NS
 810  LPRINT ID$(I); TAB(2Ø)"$";:
      LPRINT USING F$;W(I);:
      LPRINT TAB(4Ø)EC(I); TAB(54)"$";:
      LPRINT USING F$;EW(I)
 820  NEXT I
 830 LPRINT TAB(21)"------"; TAB(55)"------"
 840 LPRINT "TOTALS"; TAB(2Ø)"$";:
     LPRINT USING F$;GI;:
     LPRINT TAB(54)"$";:
     LPRINT USING F$;GW
 850 INPUT "PRESS ENTER TO CONTINUE";A$
 860 :
     '
 870 :
     ' INPUT ADDITIONAL TAX INFORMATION
 880 :
     '
 890 CLS :
     PRINT "ADDITIONAL TAX INFORMATION":
     PRINT
 900 IF IT = Ø
      THEN
       97Ø
 910 IF DI = Ø
      THEN
       PRINT "STANDARD DEDUCTION":
      ELSE
       PRINT "ITEMIZED DEDUCTIONS = $";DI
 920 PRINT "TAX CREDITS = $";TC
 930 PRINT "TAX LOSSES = $";LT
 940 PRINT AE;"EXEMPTIONS"
 950 PRINT :
     INPUT "IS THIS CORRECT";A$
 960 IF LEFT$(A$,1) = "Y"
      THEN
       1Ø1Ø:
      ELSE
       CLS
 970 INPUT "ITEMIZED DEDUCTIONS ( Ø FOR STANDARD DEDUCTION)";DI
 980 INPUT "TAX CREDITS";TC
 990 INPUT "TAX LOSSES (INPUT AS +)";LT
1000 INPUT "TOTAL NUMBER OF EXEMPTIONS";AE
1010 IF P = Ø
      THEN
       1Ø8Ø
1020 LPRINT :
     LPRINT :
     LPRINT "ADDITIONAL TAX INFORMATION :":
     LPRINT STRING$(26,"-"):
     LPRINT
1030 IF DI = Ø
```

Program continued

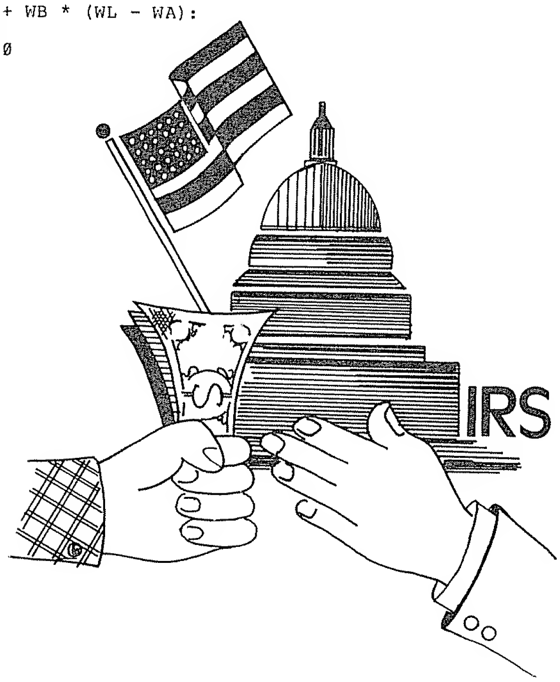
```
      THEN
       LPRINT "   STANDARD DEDUCTION":
      ELSE
       LPRINT "   ITEMIZED DEDUCTIONS = $";DI
1040 LPRINT "   TAX CREDITS = $";TC
1050 LPRINT "   TAX LOSSES  = $";LT
1060 LPRINT "   ";AE;" EXEMPTIONS"
1070 :
     '
1080 :
     ' COMPUTE TAX LIABILITY
1090 :
     '
1100 DE = DI - SD(FS):
     IF DE < 0
      THEN
       DE = 0
1110 TI = GI - DE - LT - AE * EX:
     GOSUB 1890:
     TX = INT(TX - TC)
1120 :
     '
1130 :
     ' PRINT WITHHOLDING TABLES
1140 :
     '
1150 FOR I = 1 TO NS:
      IF EW(I) = 0
       THEN
        1320
1160  WH = GW - EW(I) + TW(I):
      WI = 52 * RW(I) - EC(I) * EX:
      GOSUB 1970:
      CW = WW
1170  CLS :
      PRINT "TAX LIABILITY = $";TX; TAB(43)TB * 100;" % BRACKET"
1180  PRINT "  FOR INCOME FROM ";ID$(I)
1190  PRINT TAB(5)"EXEMPTIONS"; TAB(22)"ESTIMATED REFUND(+)";
      TAB(44)"CHANGE IN TAKE-HOME"
1200  PRINT TAB(7)"CLAIMED"; TAB(23)"OR BALANCE DUE(-)"; TAB(47)"PAY
      PER WEEK"
1210  IF P = 0
       THEN
        1260
1220  LPRINT :
      LPRINT :
      LPRINT "TAX LIABILITY = $";TX; TAB(43)TB * 100;" % BRACKET"
1230  LPRINT :
      LPRINT "  FOR INCOME FROM ";ID$(I)
1240  LPRINT :
      LPRINT TAB(5)"EXEMPTIONS"; TAB(22)"ESTIMATED REFUND(+)";
      TAB(44)"CHANGE IN TAKE-HOME"
1250  LPRINT TAB(7)"CLAIMED"; TAB(23)"OR BALANCE DUE(-)"; TAB(47)"PAY
       PER WEEK":
      LPRINT STRING$(63,"-")
1260  FOR E = 0 TO 10:
       WI = 52 * RW(I) - E * EX:
       GOSUB 1970
1270   R = WH + WW * WR - TX
1280   PRINT TAB(9)E; TAB(28)"$";:
       PRINT USING F$ + "-";R;:
       PRINT TAB(49)"$";:
       PRINT USING F$ + "-";CW - WW
1290   IF P = 1
        THEN
         LPRINT TAB(9)E; TAB(28)"$";:
         LPRINT USING F$ + "-";R;:
         LPRINT TAB(49)"$";:
         LPRINT USING F$ + "-";CW - WW
1300   NEXT E
```

```
1310  INPUT "PRESS ENTER TO CONTINUE";A$
1320  NEXT I
1330 :
     '
1340 :
     '   SAVE DATA TO TAPE
1350 :
     '
1360 CLS :
     INPUT "SAVE DATA TO TAPE";A$
1370 IF LEFT$(A$,1) = "N"
      THEN
       END
1380 PRINT "PREPARE CASSETTE, PRESS ENTER":
     INPUT A$
1390 TE = NS:
     NS = 0:
     FOR I = 1 TO TE:
      IF W(I) > 0
       THEN
        NS = NS + 1:
        NEXT I
1400 PRINT # - 1,NS,DI,TC,LT,AE,M,D,Y
1410 FOR I = 1 TO TE:
      IF W(I) = 0
       THEN
        1440
1420  PRINT # - 1,W(I),ID$(I),RW(I),EC(I),TW(I)
1430  NEXT I
1440 IF P = 1
      THEN
       FOR J = 1 TO 8:
        LPRINT :
        NEXT
1450 END
1460 :
     '
1470 :
     '   INPUT INCOME DATA FROM THE KEYBOARD
1480 :
     '
1490 W(I) = 0:
     RW(I) = 0:
     TW(I) = 0:
     EC(I) = 0:
     EW(I) = 0
1500 CLS :
     PRINT "FOR INCOME FROM ";ID$(I)
1510 INPUT "GROSS INCOME TO DATE";W(I)
1520 INPUT "ESTIMATED INCOME FOR REMAINDER OF YEAR ($/WK)";RW(I):
     W(I) = W(I) + RW(I) * WR
1530 INPUT "TAX WITHHELD TO DATE";TW(I)
1540 IF TW(I) = 0
      THEN
       RETURN
1550 INPUT "CURRENT EXEMPTIONS CLAIMED";EC(I)
1560 WI = 52 * RW(I) - EC(I) * EX:
     GOSUB 1970:
     EW(I) = TW(I) + WW * WR
1570 RETURN
1580 :
     '
1590 :
     '   COMPUTE FACTOR F FOR DAYS BETWEEN DATES
1600 :
     '   INPUTS : MM MONTH, DD DAY, YY YEAR
1610 :
     '
1620 TE = 365 * YY + 31 * MM + DD - 31
1630 IF MM < 3
```

Program continued

```
       THEN
        YY = YY - 1:
       ELSE
        TE = TE - INT(.4 * MM + 2.3)
1640 F = TE + INT(YY / 4) - INT(.75 + INT(YY / 100) * .75)
1650 RETURN
1660 :
     '
1670 :
     '  INITIALIZE STANDARD DEDUCTIONS AND EXEMPTION
1680 :
     '
1690 SD(0) = 3400:
     SD(1) = 2300:
     SD(2) = 1700:
     EX = 1000:
     RETURN
1700 :
     '
1710 :
     '  WITHHOLDING RATE OUT TABLES
1720 :
     '          - MARRIED
1730 :
     '
1740 DATA 2400,0,6600,.15,10900,.18,15000,.21,19200,.24,23600,.28,289
     00,.32,999999,.37
1750 :
     '
1760 :
     '          - SINGLE
1770 :
     '
1780 DATA 1420,0,3300,.15,6800,.18,10200,.21,14200,.26,17200,.30,2250
     0,.34,999999,.39
1790 :
     '
1800 :
     '  TAX RATE TABLES
1810 :
     '          - MARRIED FILING JOINTLY
1820 :
     '
1830 DATA 3400,0,5500,.14,7600,.16,11900,.18,16000,.21,20200,.24,2460
     0,.28,29900,.32,35200,.37,45800,.43,60000,.49,85600,.54,109400,.
     59,162400,.64,215400,.68,999999,.70
1840 :
     '
1850 :
     '          - SINGLE
1860 :
     '
1870 DATA 2300,0,3400,.14,4400,.16,6500,.18,8500,.19,10800,.21,12900,
     .24,15000,.26,18200,.30,23500,.34,28800,.39,34100,.44,41500,.49,
     55300,.55,81800,.63,108300,.68,999999,.70
1880 :
     '
1890 :
     '  ESTIMATE TAX LIABILITY (TX) FOR TAXABLE INCOME (TI)
1900 :
     '
1910 TX = 0:
     TA = 0:
     RESTORE :
     FOR J = 1 TO 16:
      READ TL,TB:
      NEXT
1920 IF FS = 1
      THEN
       FOR J = 1 TO 16:
```

```
     READ TL,TB:
     NEXT
1930 F = 1:
     IF FS = 2
      THEN
       F = 2
1940 READ TL,TB:
     TL = TL / F:
     IF TI < TL
      THEN
       TX = TX + TB * (TI - TA):
       RETURN
1950 TX = TX + TB * (TL - TA):
     TA = TL:
     GOTO 1940
1960 :
     '
1970 :
     '   ESTIMATE WEEKLY WITHHOLDING (WW) FOR W/H INCOME (WI)
1980 :
     '
1990 AW = 0:
     WA = 0:
     RESTORE
2000 IF FS = 1
      THEN
       FOR J = 1 TO 8:
        READ WL,WB:
        NEXT
2010 READ WL,WB:
     IF WI < WL
      THEN
       AW = AW + WB * (WI - WA):
       WW = AW / 52:
       RETURN
2020 AW = AW + WB * (WL - WA):
     WA = WL:
     GOTO 2010
```



INTERFACE

An Automatic Cassette Tape Interface
Send and Receive RTTY in BASIC

INTERFACE

An Automatic Cassette Tape Interface

by Dana W. Zimmerli

The cassette tape port is probably the most popular I/O interface on the TRS-80. The ease of connecting to the tape port makes it convenient as a serial port for output to audio amplifiers, printers, modems, and so on, and as an input port from serial terminals, light pens, etc. Unfortunately, most people have to perpetually switch cables back and forth in order to use this port effectively. This cassette tape interface is a simple device that automatically controls the devices to which it is connected. I first recognized the need for this when I bought a light pen for my TRS-80. Installing the light pen was aggravated by a Data Dubber (see January and February 1980 issues of *80 Microcomputing*). The number of cables to deal with resembled a bowl of spaghetti, and the light pen fell into disuse because of the complicated connections needed. Then I bought the Dancing Demon program from Radio Shack and a small speaker-amplifier. These required more cable swapping! I found the solution to my dilemmas in the circuit shown in Figure 1. This circuit provides: (1) relay contact protection for the TRS-80; (2) automatic selection of input and output for the cassette recorder interface; (3) amplification for a light pen input; and (4) a nine-volt power supply for the Data Dubber, light pen, and amplifier. All of the parts are available from your local Radio Shack store except the Data Dubber (available from The Peripheral People, P.O. Box 524, Mercer Island, WA 98040).

The key element in the circuit is the CMOS quad bilateral switch at U2. This is a unique logic element that is the equivalent of four toggle switches with a logic level control. The circuit, wired as shown, selects one of two inputs (using switches connected between 1 and 2 or 10 and 11) and, at the same time, selects one of two outputs (using switches between 3 and 4 or 8 and 9). The switching is made automatic by applying control voltages to pins 5, 6, 12, and 13. As configured, the state of the recorder relay determines the inputs and outputs connected; if the relay turns on, the input and output are switched to the cassette recorder. If it is not turned on, the input and output connect to the external devices (light pens, audio amplifiers, modems, serial terminals, etc.).

For light pen applications, four of the CMOS inverter stages are wired as a linear amplifier. The specific values of resistors determine the gain of the amplifier. In this circuit, a gain of over 30 is achieved. But, two variations are possible. First, if a light pen or other input does not require amplification, the input marked Light Pen could be wired directly to pin 10 of U2. Be certain, however, to wire the unused inputs of U1 (3, 5, 7, and 14) to either ground or +9 volts. CMOS does not work properly with uncommitted (open) inputs. Secondly, though I recommend it, a Data Dubber is not absolutely necessary. If you don't want the Data Dubber (or have it connected externally), connect the line marked "to Data Dubber" to pin 1 of U2.

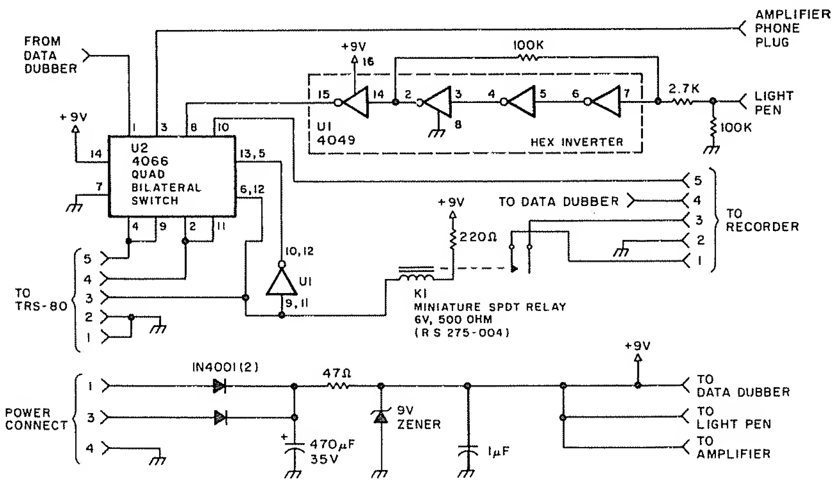


Figure 1. *Circuit diagram*

The Data Dubber connects between the recorder and the TRS-80, reshaping the pulses for higher reliability. One of the features of the circuit is an automatic turn-off if there is no data coming in. Unfortunately, if you try to load SYSTEM tapes created by the EDTASM program, you may find gaps in the data which will turn off the Data Dubber. Since the automatic turn-off isn't necessary if you have the 9-volt supply for the entire circuit, you may disable the automatic switch by removing transistor Q2 and putting a jumper between the emitter and collector.

Construction of this interface is possible with point-to-point, wire-wrap, or printed circuit board—in short, use your favorite style since layout is not critical. Figure 2 shows the parts layout for my unit; I used wire-wrap on perforated board. The power supply is driven from an ac voltage similar to the TRS-80 supply. The pins indicated out on the circuit diagram correspond to the pins on the DIN plug for the TRS-80 supply. An alternative is to obtain an 18-volt ac, 1-Amp transformer with a center-tap. The center-tap connects to pin 4 (ground), and the 18-volt ac connects to pins 1 and 3.

A coaxial power connector delivers power for an amplifier or other external device. The jack for the amplifier must be grounded to the barrel or long part of the plug. My light pen plug is a three-conductor phone jack, so I have wired ground, +9 volts, and a signal into that same jack.

The TRS-80 is connected to the recorder with five-pin DIN receptacles. When you connect the existing recorder cable to these receptacles, you might find that the plastic sleeve around the plug is too large. If so, cut off the plastic plug and replace it with another plug. Wire pin 5 to the tip of the AUX jack (gray); pin 4 to the tip of the EAR jack (black); pin 2 to the barrels of these two jacks; and pins 1 and 3 to the smaller MIC jack (small gray jack). It doesn't matter which part of the small MIC jack goes to which pin (1 or 3).

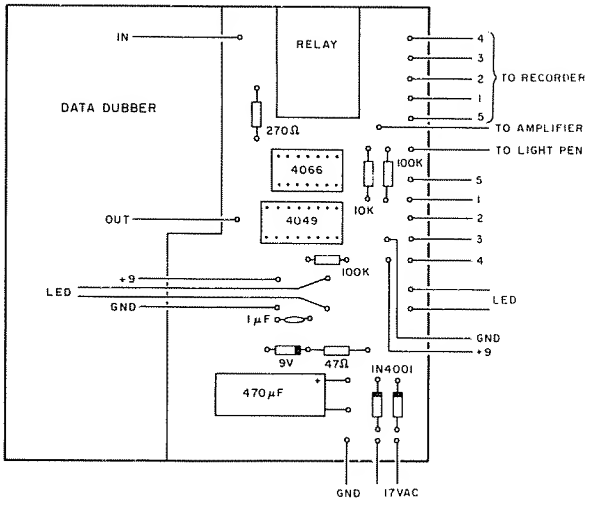


Figure 2. *Parts layout*

Remember that the recorder relay determines the input and output devices. If you use the tape recorder, then the relay must be turned on by issuing an OUT 255,4 instruction. This is the instruction used by CLOAD and CSAVE to load and save tapes, and by PRINT#-1 and INPUT#-1 to use the tape. If you use the other devices, make certain that the program doesn't turn on the relay. Radio Shack's Dancing Demon does not turn on the relay; so it doesn't require any modifications. Light pen software may or may not turn on the relay; so you will have to check to see if yours does.

That's all it takes to end your cable-swapping woes. Just turn on your computer and you'll be ready to use any device connected to the interface.

Send and Receive RTTY in BASIC

by Louis C. Graue K8TT

This article contains a circuit diagram for an interface and a BASIC program which enable your computer to send and receive RTTY. The program provides features that make operating a pleasure. You need to enter a station only once in order to send an automatic ID at any time, and you can send prepared messages of any length with the push of a key. You can also set up a prompt to ask for changes of variables within the prepared message. Best of all, since the program is in BASIC, you can easily alter or expand it to suit your own style of operation. Table 1 lists the parts you will need.

Number	Type	+5	Ground
02_1	74LS02	14	7
02_2	74LS02	14	7
04	74LS04	14	7
30	74LS30	14	7
367	74LS367	16	8
UART	TR1602B	see schematic	
555	NE555V	see schematic	

Table 1. *Parts list*

Operating the Program

You can test the program without the interface for practice in operating it before you go on the air. You need to simulate reception by making a temporary addition of one line to the program. Key in the program and enter the temporary line as follows:

```
101 A=49
```

Type RUN and press ENTER. The following should appear at the bottom of the screen:

```
------------------  RECEIVE MODE  ------------------
""""""""""""""""""""""""""""""""""""""""""""""""""""""""
```

The quotation marks flow across the screen to simulate the message being received.

Next, hold the SHIFT key down and press T. At the bottom of the screen, the following will appear:

```
------------------  TRANSMIT MODE  ------------------
```

Notice that the received message remains visible so that you can refer to it while typing a response. Type a few letters, and they should appear at the bottom. They will be transmitted as you type.

Now hold the SHIFT key down and press the letter I. The following will appear:

```
CALL/NAME/QTH OF CONTACT?____
```

Answer by typing in KA0BVW (BILL) IN TOPEKA, then press ENTER. This is the only time you will have to enter the call for any future ID. Press SHIFT D and watch the ID message print at the bottom of the screen. It should appear as follows:

```
KA0BVW (BILL) IN TOPEKA DE K8TT (LOU) IN BOWLING GREEN, OHIO
```

Finally, test the station information message by holding down the SHIFT key and pressing B (for brag). A neatly formatted message describing the station will print as you sit and watch the computer do the work.

Explanation of the Program

Line 10 clears string space. You must increase the number when an out of string space (OS) error appears. Line 20 turns on the cursor and dimensions the variables.

Lines 30–40 contain lookup tables to translate between ASCII and Baudot. The read data method cannot handle a quotation mark, so the special assignment for A$(49) is necessary.

In line 50, R keeps track of the letters-figures shifting. R = 0 for letters and R = 32 for figures. It is initialized to letters. Line 60 prints the receive mode display.

Lines 70–90 check the UART (Universal Asynchronous Receiver/Transmitter) for available data and the keyboard for a SHIFT T. The latter causes a change to the transmit mode. Line 100 places incoming data in variable A. Line 110 intercepts figure or letter shifts and fixes R. There is an automatic change to letters in line 115 after it receives a space. Line 120 translates Baudot to ASCII and prints the character.

Line 300 prints the transmit mode display, and line 310 changes keyboard entries to ASCII. Line 400 intercepts uppercase entries and directs the program to the appropriate subroutine. You should put new entries to call additional messages after this line.

Line 410 intercepts carriage return, line feed, and space for decoding and action. Line 430 discards illegal characters or sends the program to the letters routine.

Figures enter in line 450; variable T keeps track of the figures-letters shift. If necessary, T is set to 1, and a figure shift is transmitted. Line 460 translates figure to Baudot and calls the transmit subroutine.

Letters enter at line 470; T is changed to a 0 if necessary, and a letters shift is transmitted. Line 480 translates letter to Baudot and calls the transmit subroutine. Line 600 waits for the UART to accept the character, and line 610 sends the character.

Lines 700–710 transmit and display any string set equal to M$. They also handle all prepared messages.

Line 800 sends a CQ message when it is called. Type in your own version between the quotation marks. The CHR$(10) and CHR$(13) send a carriage return and line feed for formatting.

Line 900 asks for input of the call sign of a contact. The second half of the ID message is in line 920. Type in your own message between the quotation marks, leaving a space after the first quotation mark.

Line 950 combines the inserted call with your fixed call for the ID message, then sets it equal to M$ so the 700 routine can send it.

The station information message is found in lines 1000–1050. M$ cannot be longer than 255 characters. To send a long message, split it into parts. Notice how the CR and LF (CHR$(10) and CHR$(13)) are inserted so that the message will be neatly formatted. The data must be entered in lines 2000–2030 exactly as printed, or the translations will be a mess.

Modifications

Suppose you want to send the message "Merry Christmas and a Happy New Year" without typing it out each time you send it. First, decide which shifted key to assign to this message. If you choose SHIFT M which has ASCII code 109, add a line between lines 400 and 410 of the program as follows:

```
401 IF X = 109 THEN 3000
```

At line 3000 type:

```
3000 M$ = "MERRY CHRISTMAS AND A HAPPY NEW YEAR.":GOSUB700:GOTO310
```

In order to have the message set off by itself in the printout, add a CR and an LF to both ends as follows:

```
3000 M$ = CHR$(10) + CHR$(13) + "MERRY - - - YEAR" + CHR$(10) + CHR$(13):
     GOSUB700:GOTO310
```

To transmit this message, hold down the SHIFT key and press M. To send a message that needs to have parts of it updated periodically, follow the example of adding call signs to the ID message in the program.

Interface

The circuit (see Figure 1) is connected between a Flesher Corporation TU-170 and the TRS-80 keyboard 40-pin connector. A 19-wire ribbon cable will make all necessary connections to the 40-pin connector. Do not forget to connect the signal ground, pin 29, to the circuit ground.

The 1k potentiometer on the clock can best be set to 727 Hz for 60 words per minute RTTY by using a frequency counter. You can also set it by tuning in a RTTY signal and adjusting until you get solid copy. It is also possible to send and receive 100 words per minute by switching in the proper value capacitor to change the clock frequency to 1760 Hz.

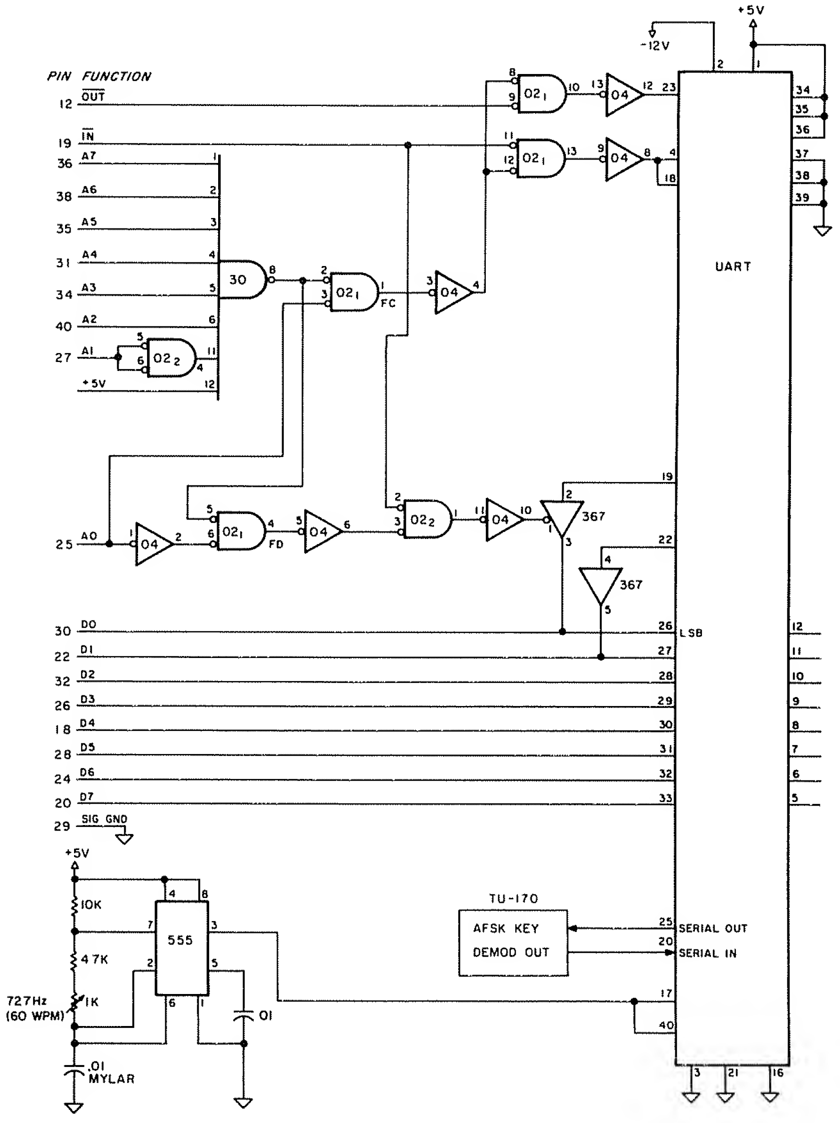


Figure 1. *Baudot serial-parallel conversion schematic*

Notice that the send and receive pins of the UART are tied together. The UART does all the timing and conversion between parallel and serial I/O. This makes programming easier and leaves the computer time to do other things.

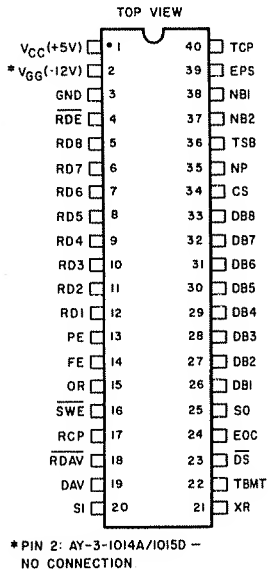


Figure 2. *UART pin configuration*

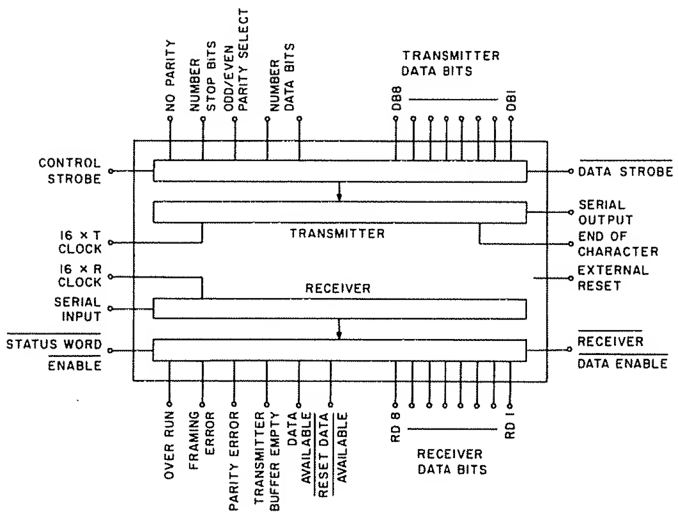


Figure 3. *UART block diagram*

On-Air Operation

To operate on the air, enter and run the program. Search the 20-meter band between 14080 and 14100 for RTTY signals. When a station calls CQ, wait until the operator signs and then press SHIFT T, followed by SHIFT I. Enter the station's call and press SHIFT D. Type in your message, and when finished, again press SHIFT D to ID. Finally, press SHIFT R to return to the receive mode.

Program Listing

Encyclopedia Loader™

```
  2 :
    ' TRANSCEIVE RTTY IN BASIC - BY K8TT - 12/21/80
  3 :
    ' ***********************************************************
  4 :
    '
  7 :
    ' SET UP TRANSLATION ARRAYS AND CLEAR STRING SPACE
  8 :
    ' """""""""""""""""""""""""""""""""""""""""""""""""""""""""""
 10 CLEAR 1000
 20 PRINT CHR$(14):
    DIM A$(63),B(60):
    DEFINT I,R,T,A,X
 30 FOR I = 0 TO 62:
     READ A$(I):
     NEXT I:
    A$(49) = CHR$(34)
 40 FOR I = 1 TO 57:
     READ B(I):
     NEXT I
 44 :
    '
 45 :
    ' RECEIVING ROUTINE
 46 :
    ' """""""""""""""""""""""""""""""""""""""""""""""""""""""""""
 50 R = 0:
    PRINT
 60 PRINT @960, STRING$(21,"-");">>> RECEIVE MODE <<<"; STRING$(22,"
    -")
 70 A = INP(253)
 80 B$ = INKEY$:
    IF B$ = ""
     THEN
      90:
     ELSE
      IF ASC(B$) = 116
       THEN
        300:
       ELSE
        70
 90 IF A = 254 GOTO 70
100 A = INP(252)
110 IF A = 27
     THEN
      R = 32:
     ELSE
      IF A = 31
       THEN
        R = 0:
        GOTO 70
115 IF A = 4
     THEN
      R = 0
120 A = A + R:
    PRINT A$(A);:
    GOTO 70
295 :
    '
296 :
    ' TRANSMITTING ROUTINE
297 :
    ' """""""""""""""""""""""""""""""""""""""""""""""""""""""""""
300 PRINT :
    PRINT @960, STRING$(21,"-");">>> TRANSMIT MODE <<<"; STRING$(21,
    "-")
```

```
310 X$ = INKEY$:
    IF X$ = ""
     THEN
      310:
     ELSE
      X = ASC(X$):
      GOSUB 400:
      GOTO 310
400 IF X < 96
     THEN
      PRINT X$;:
     ELSE
      IF X = 114
       THEN
        50:
       ELSE
        IF X = 99
         THEN
          800:
         ELSE
          IF X = 105
           THEN
            900:
           ELSE
            IF X = 100
             THEN
              950:
             ELSE
              IF X = 98
               THEN
                1000
410 IF X = 10
     THEN
      X = 2:
     ELSE
      IF X = 13
       THEN
        X = 8:
       ELSE
        IF X = 32
         THEN
          X = 4:
          GOSUB 600:
          RETURN
430 IF X > 90
     THEN
      RETURN :
     ELSE
      IF X > 64
       THEN
        470:
       ELSE
        IF X < 32 RETURN
450 IF T = 0
     THEN
      T = 1:
      X = 27:
      GOSUB 600:
      X = ASC(X$)
460 X = X - 6:
    X = B(X):
    GOSUB 600:
    RETURN
470 IF T = 1
     THEN
      T = 0:
      X = 31:
      GOSUB 600:
      X = ASC(X$)
480 X = X - 64:
```

Program continued

```
     X = B(X):
     GOSUB 600:
     RETURN
600 A = INP(253):
     IF A = 252
      THEN
       600
610 OUT 252,X:
     RETURN
697 :
    '
698 :
    ' MESSAGE SENDING ROUTINE
699 :
    ' """""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
700 Q = LEN(M$)
710 FOR L = 1 TO Q:
      X$ = MID$(M$,L,1):
      X = ASC(X$):
      GOSUB 400:
      NEXT L:
     RETURN
800 M$ = CHR$(10) + CHR$(13) + "CQ CQ CQ CQ CQ CQ CQ CQ CQ CQ CQ CQ
     CQ DE K8TT K8TT K8TT K8TT   (LOU) IN BOWLING GREEN, OHIO"
     + CHR$(10) + CHR$(13):
     GOSUB 700:
     GOTO 310
900 PRINT :
     INPUT "CALL/NAME/QTH OF CONTACT";C$
920 D$ = " DE K8TT (LOU) IN BOWLING GREEN, OHIO " + CHR$(10)
     + CHR$(13):
     RETURN
950 M$ = CHR$(10) + CHR$(13) + C$ + D$:
     GOSUB 700:
     GOTO 310
1000 A1$ = "---   A M A T E U R   R A D I O   S T A T I O N   K 8 T T
     ---"
1010 B1$ = CHR$(10) + CHR$(13)
1020 C1$ = "OPERATOR: LOUIS C. GRAUE  (LOU)"
1021 D1$ = "QTH: 624 CAMPBELL HILL ROAD, BOWLING GREEN, OHIO 43402"
1022 E1$ = "RIG: HEATHKIT - SB104A - SB230 LINEAR - TH6DXX AT 40 FEET
     "
1023 F1$ = "RTTY: TRS-80 COMPUTER - HOMEBREW PROGRAMS & INTERFACE"
1024 G1$ = "JOB: MATH PROFESSOR AT BOWLING GREEN STATE UNIVERSITY"
1025 H1$ = "HOBBIES: COMPUTERS - ELECTRONICS - GARDENING - HOMING PIG
     EONS"
1026 I1$ = STRING$(63,".")
1030 M$ = B1$ + A1$ + B1$ + C1$ + B1$:
     GOSUB 700
1040 M$ = D1$ + B1$ + E1$ + B1$ + F1$ + B1$:
     GOSUB 700
1050 M$ = G1$ + B1$ + H1$ + B1$ + I1$ + B1$:
     GOSUB 700:
     GOTO 310
2000 DATA ,E," ",A," ",S,I,U," ",D,R,J,N,F,C,K,T,Z,L,W,H,Y,P,Q,O,B,G,
      ,M,X,V, , ,3," "
2010 DATA -," "," ",8,7," ",$,4,"'",",",!,":",(,5, ,),2,#,6,0,1,9,?,&
     , ,.,/,";"
2020 DATA 3,25,14,9,1,13,26,20,6,11,15,18,28,12,24,22,23,10,5,16,7,30
     ,19,29,21,17
2030 DATA 13,17,20,9,0,26,11,15,18,0,0,12,3,28,29,22,23,19,1,10,16,21
     ,7,6,24,14,30,0,0,0,25
```

TUTORIAL

A Better Way

by Robert V. Meushaw

I'm sure many of you have been faced with situations in which the program you were writing just wouldn't perform to your expectations because of limitations which seemed beyond your control. In some cases, processor speed or inefficient data storage techniques may limit your capabilities, but in almost every hopeless situation I work on, I find that, with perseverance, there is always a better way.

One of the most important lessons I have learned while programming is that you should use your computer to help you analyze and resolve your programming problems. In this chapter I will describe several programming applications I have faced with my Level I TRS-80 and the approaches that I used to investigate such limitations.

Problem One

Computing vector magnitudes when given the x- and y-coordinates (Figure 1) requires a square root function that is not built in to the Level

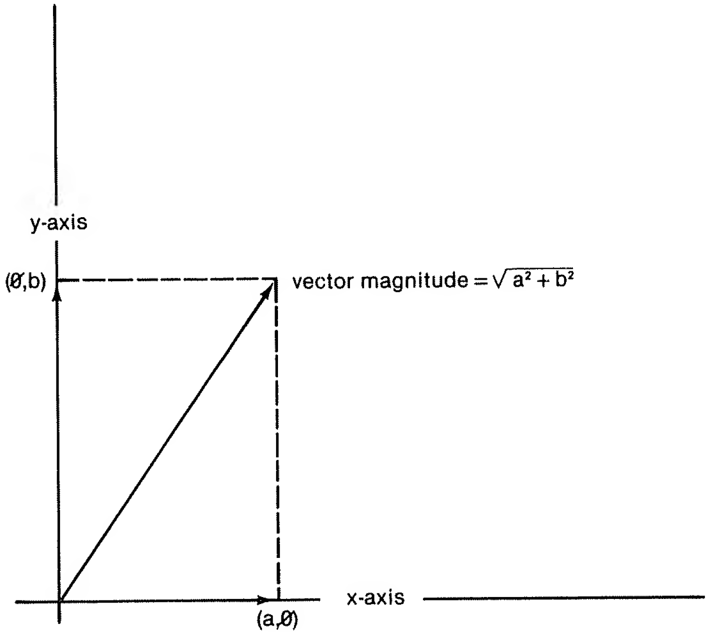


Figure 1

I. Fortunately, the appendix of the Level I manual includes a square root function (Program Listing 1), but it performs slowly. Watching the results being printed, I could almost feel the burden of the routine. I began to investigate how to improve the speed of my program.

Looking at Program Listing 1, you can see that the routine makes successive approximations to the square root of X which eventually converge (within the limits of the computer's accuracy) to the square root of X. The first approximation, Y, is taken as X/2. W keeps track of the error in the approximation, and whenever the error is zero or is the same as the previous iteration, the subroutine returns with Y as the square root of X. This seems fairly straightforward, but, as you can see, there is really quite a lot going on in one iteration, which is reason enough for its lack of speed.

Wondering how many iterations of such a routine are necessary to compute a square root, I decided to make some tests. The test program is shown in Program Listing 2. It generates random numbers to be input into the square root subroutine. Each time a new number is generated, a counter is set to zero. The square root subroutine is called and modified to increment the counter with each iteration. When the subroutine returns, C contains the number of iterations necessary to compute the square root of X. The value of C is used to increment the array element A(C). Using this technique, the array element A(i) keeps count of the number of values of X which require iterations in the square root subroutine to return an answer.

The first three lines in the program initialize the array elements to zero. Line 50 determines the value of X. This line is not fully specified in Program Listing 2, because I ran the program a number of times with different expressions for X. In each case the program examines 15,000–20,000 values of X. This often requires several hours of run time. (On several occasions I let the test run overnight.)

Usually, when a reasonable number of samples has been examined, I interrupt the program and execute the routine beginning on line 2000, which displays the percentage of samples requiring iteration counts of one to one hundred.

Some Interesting Results

The program is first run with X = RND(0). This sets X equal to a random number between 0 and 1. The resulting iterations are shown in Table 1. Beside each iteration value is its percentage. We can see that 34.6 percent of the numbers require six iterations, 48 percent require seven iterations, and so on. The highest number of iterations recorded is fourteen, and no number requires five or less.

Number of Iterations	Percent of Numbers Tested
< =5	0
6	34.6
7	48
8	13
9	3.3
10	.87
11	.16
12	.056
13	.01
14	.004
> = 15	0

Table 1. *Distribution of square root iteration counts for X = RND(0).*

Table 2 shows the results using X = 1/RND(30000). The range of iterations is between seven and fourteen, with the largest percentage of the numbers requiring more than twelve.

Number of Iterations	Percent of Numbers Tested
< =6	0
7	.03
8	.04
9	.33
10	1.2
11	5.4
12	19.3
13	47.8
14	25.9
> = 15	0

Table 2. *Distribution of square root iteration counts for X = 1/RND(30000).*

Table 3 shows the results of X = RND(30000). These results appear similar to those of Table 2 except that the range of iterations is offset slightly.

Number of Iterations	Percent of Numbers Tested
< =3	0
4	.004
5	.02
6	.076
7	.3
8	1.15
9	4.85
10	19
11	47.4

Table continued

12	27.2
> = 13	0

Table 3. *Distribution of square root iteration counts for X = RND(30000).*

The last set of results, shown in Table 4, is the result of using X = RND(0)*1E10. I chose this to give some very large values for X. Again, the results appear similar to those in Table 2 with a larger offset than in Table 3.

Number of Iterations	Percent of Numbers Tested
< = 14	0
15	.05
16	.17
17	.95
18	3.51
19	14.12
20	47.1
21	34.1
> = 22	0

Table 4. *Distribution of square root iteration counts for X = RND(0)*1E10.*

The Search Begins

Now that I had a reasonable understanding of how the square root subroutine works and why it consumes so much time, the problem remained to find a technique to reduce the time.

My first thought was to investigate other methods of computing the square root. One technique, which often yields good results, is a power series approximation. These approximations are used in calculating sine, cosine, natural log, and many other functions. The advantage of such a technique is the elimination of the numerous iterations the computation requires—thus saving time. Since it has been many years since I encountered such approximation techniques, I found myself digging out college books, which I had hoped never to see again. Though it was not as clear to me now how a Taylor Series or Maclaurin Series approximation works, I eventually convinced myself that there is no simple expansion that can be used to solve this problem.

As is clear from Tables 1–4, larger numbers require more iterations to compute the square root. Part of the reason for this is that the first approximation to the square root is taken as X/2, which becomes a worse approximation as X increases or decreases. Again, I considerd a power series approximation such as:

```
IF (X< = 10) THEN Y = X/2
IF (X>10) AND (X< = 100) THEN Y = X/4
IF (X>100) AND (X< =1000) THEN Y = X/20
```

I experimented with several variations of this form, but none yielded a significant reduction in the number of iterations.

A Glimmer of Light

Finally, I realized that I needed to look for a solution to my specific application and not a general one. Looking back at Figure 1, I noticed that I was not trying to find the square root of just any number, I was trying to find the length of the hypotenuse of a right triangle—knowing the length of each side! At this point I had found the first approximation—the sum of the lengths of the sides of the triangle (i.e., A + B).

My next task was to determine how good my approximation method really was. Program Listing 3 shows the program which I used to generate the lengths of each side of the right triangle and the number whose square root I needed. As in the previous set of test cases, program statements 50 and 51 are incomplete because I used four sets of values to test the same conditions as before.

The program was run using A = RND(0) and B = RND(0). The value of X was computed as (A*A + B*B). Table 5 shows the result of using the new first approximation for the square root. Compared to Table 1, the

Number of Iterations	Percent of Numbers Tested
< =2	0
3	1.1
4	13
4	13
5	52.3
6	33.6
> =7	0

Table 5. *Distribution of square root iteration counts using A = RND(0) and B = RND(0) and modified first approximation.*

Number of Iterations	Percent of Numbers Tested
< =2	0
3	1.1
4	12.7
5	52.1
6	34.1
> =7	0

Table 6. *Distribution of square root iteration counts using A = RND(30000) and B = RND(30000) and modified first approximation.*

number of iterations is significantly reduced. In this case, most numbers require five or fewer iterations, whereas Table 1 shows that most numbers require seven or more iterations.

Table 6 shows the results of the new approximation when A = RND(30000) and B = RND (30000). Notice that the results are remarkably similar to those obtained in Table 5.

Number of Iterations	Percent of Numbers Tested
1	0
2	.01
3	.96
4	13.16
5	51.6
6	34.27
> =7	0

Table 7. *Distribution of square root iteration counts using A = 1/RND(30000) and B = 1/RND(30000) and modified first approximation.*

Table 7 shows the results obtained using A = 1/RND(30000) and B = 1/RND(30000), and Table 8 shows the results obtained using A = RND(0)*1E10 and B = RND(0)*1E10.

Number of Iterations	Percent of Numbers Tested
< =2	0
3	1
4	13.5
5	51.3
6	34.2
> =7	0

Table 8. *Distribution of square root iteration counts using A = RND(0)*1E10 and B = RND(0)*1E10 and modified first approximation.*

I was astounded that the results obtained using the new approximation were almost identical in the four cases that I tested. I noticed two other important facts. Most of the numbers required five or six iterations, and no numbers required more than six iterations. This second fact is most important, since it tells me that I can construct a new square root routine eliminating the logical tests necessary to determine completion of the routine.

Referring to Program Listing 1, I can remove line 30080 by changing the basic structure of the routine to a FOR-NEXT loop using six iterations. Program Listing 4 gives the resulting square root routine. In this version the speed of the routine is increased. It seems wasteful to perform six iterations when some numbers require less, but looking at the statistics of Tables 5–8 you can see that 85 percent of the numbers require five or six iterations. Therefore, the instances of inefficiency are small.

Now that I was over the major hurdle, I wanted to make my computations even more efficient. My primary target was the parentheses which appear in line 120 of Program Listing 4. First, I experimented with alternate methods of writing the approximation equations. Figure 2 shows some of the equation derivations. The first equation shows the value of the second approximation to the square root of X. In this equation, Y2 is the second approximation and Y1 is the first. As you can see, Y2 can be simply computed from Y1.

$$Y2 = Y1 + \frac{1}{2}\left(\frac{X}{Y1} - Y1\right) = \frac{1}{2}\left(\frac{X}{Y1} + Y1\right)$$

$$Y3 = Y2 + \frac{1}{2}\left(\frac{X}{Y2} - Y2\right) = \frac{1}{2}\left(\frac{X}{Y2} + Y2\right)$$

$$= \frac{1}{2}\left[\frac{X}{\frac{1}{2}\left(\frac{X}{Y1} + Y1\right)} + \frac{1}{2}\left(\frac{X}{Y1} + Y1\right)\right]$$

$$= \frac{X}{\frac{X}{Y1} + Y1} + \frac{1}{4}\left(\frac{X}{Y1} + Y1\right)$$

Figure 2. *Equations for the second and third approximations to the square root of X*

In the next approximation, Y3 is simply expressed in terms of Y2. In equation 3, the substitution is made for Y2. This equation reveals something very interesting. If we assign the value of (Y1 + X/Y1) to Z, then two successive

$$Z = Y + X/Y$$
$$Y = Z/4 + X/Z$$

Figure 3. *Equations which combine two successive approximations to the square root of X*

approximations can be simply computed using the equations shown in Figure 3.

The fact that we have condensed two approximations into this set of equations allows us to rewrite the square root routine as shown in Program Listing 5. Notice that only three iterations of the FOR-NEXT loop are required and that the parentheses are no longer included.

My final modification to increase the speed of the routine is shown in Program Listing 6. Here, the FOR-NEXT loop is replaced with three sets of equations which compute six approximations. This straight-line coding requires more bytes of storage than the previous routine,but is slightly faster.

Program Listing 7 tests the various modifications which I have developed by timing them. Five subroutines are used for the various tests. The first subroutine is an immediate RETURN. I use this to determine how much time the main routine requires. The four other routines are the normal square root routines from the Level I manual: the normal routine, the routine using a modified first approximation and a FOR-NEXT loop, the routine using the modified approximation equation, and the routine using the straight-line coding of the approximation equations.

The results of the test are shown in Table 9. After subtracting the five-second time of the main routine from the individual test times, the subroutine times can be compared. Execution speed is cut by a factor of more than eleven in going from the normal square root routine to my final specialized version. An improvement of this magnitude makes the work involved seem well worth the trouble.

Subroutine Used	Total Time	Total Subroutine Time
Immediate RETURN	5 sec.	—
Normal square root	107 sec.	102 sec.
Program Listing 4 routine	26 sec.	21 sec.
Program Listing 5 routine	16 sec.	11 sec.
Program Listing 6 routine	14 sec.	9 sec.

Table 9. *Execution times of the various square root routines.*

Problem Two

The second example I will discuss involves an application in which my primary concern is program space and accuracy. Because the TRS-80 is limited to slightly more than 3500 bytes of usable storage, some programs can become pretty cramped. My problem was to construct an array containing values of sin(X) over a range of 0 to $\pi/2$ (i.e., $0°–90°$). As in the case of the square root function, Level I BASIC does not contain the sine function. So, again I turned to the Level I manual's sine function subroutine. This routine is shown in Program Listing 8.

The basic sine routine calculation is performed using a power series approximation. Many mathematics text books will show that sin(X) can be expressed as the infinite series:

$$\sin(X) = X - (X^{**}3)/3! + (X^{**}5)/5! - (X^{**}7)/7! + (X^{**}9)/9! - \ldots$$

In the case of Program Listing 8, the power series is computed using terms up to X**9.

The Approach

When it became apparent that I needed a few more bytes of storage for data, I examined the sine subroutine to see how many bytes I could squeeze out. Because this is a repetitive task, a FOR-NEXT loop seemed to be the appropriate method for computing the terms of the power series. After several tries, I arrived at the routine shown in Program Listing 9.

The variable Q generates the terms of the power series. Each successive value of Q is computed by multiplying the previous value by −X**2 and dividing by the product of J, the loop index and (J-1). The variable Y keeps a running sum of the power series terms. The savings in terms of storage amounts to about 25 bytes, which is enough to give me the breathing room I need. In fact, more savings are possible if unnecessary lines in the resulting sine routine are eliminated.

There is an added benefit to the modified sine routine. It is very simple and costs nothing in terms of storage to obtain more accuracy. It is only necessary to replace the nine in line 120 of Program Listing 9 with the value of the highest power term desired in the approximation. For example, 15 can be used.

The Final Test

Just to satisfy my curiosity, I decided to try the routine with more terms. I first tried using 11 as the highest power term. The test routine is shown in Program Listing 10. It uses the parameter N to specify for the sine routine the highest power term in the approximation. For each value of X, the sine routine is called twice—once using terms up to X**9 and once using terms up to X**11. The values obtained using each approximation are printed, and in addition, the two values are compared and a message is printed when the values are different.

The results of this test are both interesting and confusing. The approximation using the X**11 term is more accurate, as I had hoped; however, some results which appear to be the same when printed, produce a message which says that they are different.

I took this to mean that the internal binary representations of the values are slightly different. A portion of the results of the test is shown in Table 10. The confusing part of the results appears in the last computed value. The printed value displays 1.0000008, which contains more digits than I have ever been able to print for a number (the normal number of digits is six). Next, I tried adding another term to the power series approximation (i.e., (X**13)/13!) but it produced no further improvement in accuracy.

Value of SIN(X) Using X**9 Term	Value of SIN(X) Using X**11 Term	Test Result
.903989	.903898	different
.923880	.923880	different
.941544	.941544	different
.956941	.956941	different
.970032	.970031	different
.980786	.980785	different
.995186	.995185	different
.998798	.998795	different
1.000000	1.0000008	different

Table 10. *Partial listing of results from test program shown in Program Listing 10. Note that some values are not equal in a logical equivalence test even though the printed values are the same.*

Aside from these few puzzling occurrences, which I have yet to fully resolve, the results of my experiments have been successful. I have achieved a savings in program space and an increase in accuracy.

The Moral

I hope that the examples that I have presented will encourage you to find new and better ways to program routines in your applications. Don't assume that the method someone else uses is necessarily the best way for you. I'm sure that you too will discover that finding a better way is perhaps one of the most rewarding aspects of programming. Most important of all, however, remember to make use of all your resources—including your computer—to improve your work.

Program Listing 1

TRS-80 Level I manual square root subroutine (shorthand notation has been expanded).

```
30010 REM  * SQUARE ROOT*  INPUT X, OUTPUT Y
30020 REM  ALSO USES W AND Z INTERNALLY
30030 IF X = 0
       THEN
        Y = 0:
        RETURN
30040 IF X > 0
       THEN
        30060
30050 PRINT "ROOT OF NEGATIVE NUMBER?":
      STOP
30060 Y = X * .5:
      Z = 0
30070 W = (W / Y - Y) * .5
30080 IF (W = 0) + (W = Z)
       THEN
        RETURN
30090 Y = Y + W:
      Z = W:
      GOTO 30070
```

Program Listing 2

*Program used to investigate the operation of the square root subroutine. The program was run with the value of X in line 50 given as X = RND(0), X = 1/RND(30000), X = RND(30000), and X = RND(0)*1E10. The program starting at line 2000, run after a sufficient number of samples were taken, prints the percentage of numbers for each iteration count.*

```
  10 FOR I = 1 TO 100
  20  A(I) = 0
  30  NEXT I
  40 J = 1
  50 X = (... ON E OF SEVERAL STATEMENTS...)
  60 C = 0
  70 GOSUB 1000
  80 A(C) = A(C) + 1
  90 PRINT J; "VALUES HAVE BEEN USED"
 100 J = J + 1
 110 GOTO 50
 120 END
 130 :
     :
 140 :
     :
1000 Y = X * .5:
     Z = 0
1010 W = (X / Y - Y) * .5
1020 C = C + 1
1030 IF (W = 0) + (W = Z)
      THEN
       RETURN
1040 Y = Y + W:
     Z = W:
     GOTO 1010
1050 :
     :
1060 :
     :
2000 T = 0
2010 FOR I = 1 TO 100
2020  T = T + A(I)
```

Program continued

```
2030  NEXT I
2040 FOR K = 1 TO 100
2050  PRINT K, 100 * A(K) / T
2060  NEXT K
```

Program Listing 3

Program used to compute the square root of X using the first approximation Y = A + B.

```
  10 FOR I = 1 TO 100
  20  A(I) = 0
  30  NEXT I
  40 J = 1
  50 A = (... VARIOUS EXPRESSI ON S ...)
  51 B = (... VARIOUS EXPRESSI ON S ...)
  60 X = A * A + B * B
  70 C = 0
  80 GOSUB 1000
  90 A(C) = A(C) + 1
 100 PRINT J; "VALUES HAVE BEEN USED"
 110 J = J + 1
 120 GOTO 50
 130 END
 140 :
 150 :
1000 Y = A + B:
     Z = 0
1010 W = (W / Y - Y) * .5
1020 C = C + 1
1030 IF (W = 0) + (W = Z)
      THEN
       RETURN
1040 Y = Y + W:
     Z = W:
     GOTO 1010
```

Program Listing 4

Modified square root routine using first approximation and FOR-NEXT loop.

```
100 Y = A + B:'                  COMPUTE APPROX. TO SQ. RT.
110 FOR K = 1 TO 6:'             SET LOOP FOR SIX ITERATIONS
120     W = (W / Y - Y) * .5:'   COMPUTE EROR
130     Y = Y + W:'              COMPUTE NEW APPROX.
140 NEXT K:'                     LOOP
150 RET.
```

Program Listing 5

Revised square root routine using new first approximation and combined approximation equations. Note that only three loop iterations are required.

```
100 Y = A + B:'                  COMPUTE APPROX. TO SQ. RT.
110 FOR K = 1 TO 3:'             SET LOOP FOR THREE ITERATIONS
120     Z = Y + X/Y:'            COMPUTE TWO APPROXIMATIONS
130     Y = Z/4 + X/Z:'            USING MODIFIED EQUATIONS
140 NEXT K:'                     LOOP
150 RET.
```

Program Listing 6

Final square root routine which eliminates FOR-NEXT loop to maximize speed.

```
100 Y = A + B:'                     COMPUTE APPROX. TO SQ. RT.
110 Z = Y + X/Y:'                     USING THREE SETS OF
120 Y = Z/4 + X/Z:'                   MODIFIED EQUATIONS TO
130 Z = Y + X/Y:'                     COMPUTE SIX
140 Y = Z/4 + X/Z:'                   APPROXIMATIONS
150 Z = Y + X/Y
160 Y = Z/4 + X/Z
170 RET.
```

Program Listing 7

Program used to test the efficiency of the various square root subroutines.

```
10 FOR I = 1 TO 100:'              SET LOOP FOR 100 ITERATIONS
20 A=RND(30000):B=RND(30000):'     PICK VALUES FOR A AND B
30 X = A * A + B * B:'             COMPUTE SQUARE OF VECTOR MAG
40 GOSUB 100:'                     COMPUTE SQUARE ROOT
50 NEXT I:'                        LOOP
60 PRINT "TIMING TEST COMPLETED":' PRINT MESSAGE WHEN DONE
70 END
```

Program Listing 8

TRS-80 Level I manual sine function subroutine.

```
30370 REM  *SIN* INPUT X IN DEGREES, OUTPUT Y
30371 REM  ALSO USES Z INTERNALLY
30376 Z = ABS(X) / X:
      X = Z * X
30380 IF X > 360
       THEN
        X = X / 360:
        X = (X - INT(X)) * 360
30390 IF X > 90
       THEN
        X = X / 90:
        Y = INT(X):
        X = (X - Y) * 90:
        ON Y GOTO 30410,30420,30430
30400 X = X / 57.29578:
      IF ABS(X) < 2.4861E - 4
       THEN
        Y = 0:
        RET.
30405 GOTO 30440
30410 X = 90 - X:
      GOTO 30400
30420 X = - X:
      GOTO 30400
30430 X = X - 90:
      GOTO 30400
30440 Y = X - X * X * X / 6 + X * X * X * X * X / 120 - X * X
      * X * X * X * X * X / 5040
30450 Y = Y + X * X * X * X * X * X * X * X * X / 362880:
      IF Z = - 1
       THEN
        Y = - Y
30460 RET.
```

Program Listing 9

Modified sine routine. The statements of this routine could be used to replace the power series expression on lines 30440 and 30450 of Program Listing 8.

```
100 Q = X:'                    Q SET TO FIRST TERM IN PWR SERIES
110 Y = X:'                    Y IS SUM OF POWER SERIES TERMS
120 FOR J = 3 TO 9 STEP 2:'    SET LOOP TO USE TERMS UP TO X**9
130 Q = - Q * X/J * X/(J-1):'COMPUTE NEXT PWR. SERIES TERM
140 Y = Y + Q:'                ADD TERM TO SUM
150 NEXT J:'                   LOOP
160 RETURN
```

Program Listing 10

Program used to test the effect upon SIN(X) of different power series approximations.

```
 10 FOR I = 0 TO 32:'               SET LOOP FOR 32 SINE WAVES
 20 X = 1.5707963 * I/32:'          X IS INPUT (0<=X<= /2)
 30 T = X:'                         SAVE X
 40 N = 9:GOSUB 1000:'              COMPUTE SQUARE ROOT
 50 A(1) = Y:'                      SAVE ANSWER
 60 X = T:'                         RESTORE X
 70 N = 11:GOSUB 1000:'             COMPUTE NEW SQUARE ROOT
 80 A(2) = Y:'                      SAVE ANSWER
 90 PRINT A(1),A(2):'               PRINT RESULTS
 100 IF A(1)<>A(2)P."DIFFERENT":P.:'PRINT IF VALUES DIFFERENT
 110 NEXT I:'                       LOOP
 120 END
1000 Q = X:'                      COMPUTE SINE OF X
1010 Y = X:'                        USING THE HIGHEST
1020 FOR J = 3 TO N STEP 2:'        POWER TERM SPECIFIED
1030 Q = -Q * X/J * X/(J+1):'       BY THE CALLING
1040 Y = Y + Q:'                    ROUTINE
1050 NEXT J
1060 RET.
```

Don't Be a Slow POKE, Take a PEEK at Your Computer

by Hunt K. Brand

The biggest problem when writing a BASIC real-time game program (and many other programs) is speed. The two things common to most of these programs are input commands and video displays. You can speed up the way the computer accomplishes both of these functions by using the BASIC commands PEEK and POKE. I will explain in simple terms how PEEK and POKE work and how they are related to the PRINT and SET commands, along with a detailed description of the video display and how it is organized and accessed.

Faster Keyboard

The INKEY function works for single inputs only. It is also not the fastest way to access the keyboard. The most used keys in games are the arrow keys to steer and the space bar to fire, or similar control functions of that sort. There is one address you can look at, or PEEK at, to see all of these control keys: arrow keys, space bar, CLEAR, ENTER, and BREAK. If you PEEK at this address, you can see what key or what combination of these keys is being pressed (see Table 1). To get an idea of how address 14400 (decimal) works and the keys it checks, try the program in Program Listing 1.

The Video

The video display can be visualized as a 64 × 16 graph. There are 64 print locations, or addresses, across the screen (0–63) for each of the sixteen lines down (0–15). Each print location is divided into six parts—two across and three down. These parts are called pixels (picture elements). Each pixel, or any combination of pixels, can be turned on and off using the SET, RESET, PRINT, and POKE commands. These commands will be described later. 64 × 16 = 1024 total print locations on the video display, and each print location has six possible pixels, so 1024 × 6 = 6144 total different pixels on the screen. Only one letter can fit into a print location, whereas a total of six pixels will be in the same print location. Stated another way: You can have a total of 1024 letters on the screen or a total of 6144 pixels on the screen. Each print location can be thought of as a particular video memory location.

Address: 14400 decimal (3840 hex)
Keys stored at address: arrows, space bar, CLEAR, ENTER, and BREAK keys.

Keys pressed	Number stored in address 14400
Arrow keys	
up arrow	8
down arrow	16
right arrow	64
left arrow	32
up and left	40
up and right	72
down and left	48
down and right	80
Other keys	
ENTER key	1
CLEAR key	2
Space bar	128

Note: If the space bar is pressed at the same time the other keys are pressed, it adds 128 to the number.

Table 1. *Keyboard control keys*

Faster Video

PRINT (particularly PRINT@), SET and RESET, and POKE are the commands used to send information to the video display. They all use numbers to put information on the screen. The proper use of these functions can lead to a faster display process. Understanding the advantages and disadvantages of each, and how each works in comparison to the other functions can help you decide which function, or what combination of functions, to use.

The PRINT@xxxx command tells the computer to print at location xxxx on the screen, where xxxx is between 0 and 1023 decimal. The SET(X,Y) command tells the computer to turn on a particular pixel by supplying two numbers—the x- and y-axis numbers. RESET works the same way as SET, but it turns off a particular pixel. POKE xxxx,ccc puts a letter or any combination of pixels on the screen at address xxxx, where xxxx, in this example, has the range 15360 through 16383 decimal. The number ccc is the ASCII (American Standard Code for Information Interchange) or graphic code in decimal for the character you want to print. This code number is used for PRINT CHR$(ccc) or POKE xxxx,ccc. Each command will print the same character on the screen. The command POKE

15360,191 puts a solid graphic block, CHR$(191), at address 15360 decimal. The first video address is 15360, at the top left hand corner of the video display.

Display Characters

The TRS-80 has a total of 255 character codes, of which 128 are display characters. The display characters can be divided into two groups—alphanumeric and graphic. There are 64 alphanumeric character codes (letters, numbers, and punctuation) that can be displayed on the screen, and 64 graphic character codes. The alphanumeric character codes range from 32 to 127 and are referred to as ASCII code numbers. Although there are ASCII code numbers for lowercase letters (96–127), they are converted to uppercase characters (32–95) when they are printed, so in effect there are 64 alphanumeric characters. There is one graphic character for each ASCII character. The ASCII code for a space, a blank print location, is the number 32, while the graphic code number for a blank is 128. The number 32 is the first ASCII code number, and 128 is the first graphic code number. This illustrates the relationship between the two groups and how one is derived from the other.

Although you can have any combination of pixels in a print location, there is a corresponding code number for each. If you wanted each of the two top pixels of a print location (code number 131) to be on at the same time the two bottom pixels were on (code number 176), you could not put one on top of the other. You would have to use the code that already exists for that combination—code number 179.

There is a relationship between the graphic characters and their codes. The lowest graphic code number, 128, turns off all six pixels of a print location. The highest number, 191, turns all six pixels on. The next lowest number, 129, turns on the top left-hand pixel of a print location. The next highest number, 190, turns on all but the top left-hand pixel. In each case, one is the opposite of the other. This relationship continues through the whole range of graphic codes.

Program Listing 2 will give you an idea of how letters and graphics fit into a print location. The code number will be displayed as each code is POKEd. After each character, the full print location will be turned on graphically with the POKE16000,191 command. This will let you see how each character fits into a print location. Press the space bar for a new number, or hold it down for a rapid check.

Lowercase Characters

If you want to use lowercase letters, you can have your keyboard modified. Although there will be more display characters, you would still have the same number of graphic characters. A lowercase driver program is used to display the lowercase letters. This is needed because the computer normally converts all letters to uppercase; the driver program bypasses the conversion, allowing the lowercase letters to appear on the screen. If the driver program is not active, you cannot PRINT lowercase letters, but you can POKE them. If you issue the command PRINT@0,CHR$(118) without a lowercase driver, an uppercase V will be printed. If you issue a POKE 15360,118 command, a lowercase v will appear on the screen at the same place where the uppercase V appeared, even though there is no lowercase driver. The POKE command works because it puts the character directly into video memory, avoiding the conversion to uppercase that the PRINT command goes through.

POKE Explained

POKE could be loosely translated as meaning put. When you tell the computer to POKExxxxx,ccc you are telling it to put a certain number (ccc) directly into a certain memory location (xxxxx). The number ccc has the range 0 through 255. The number xxxxx is the memory address of any RAM (random access memory) location into which you want to put the number. Be very careful when using this command, because if you put a number in the wrong memory location (one used by the computer to keep track of what it is doing), it might cause the computer to freeze up.

We are concerned with POKEing the video memory addresses between 15360, the upper left corner of the video display, and 16383, the bottom right corner of the video display. If you visualize the video display as a window which allows you to look at a certain section of RAM, it will help you understand how POKE works. Each time you POKE a number into a video memory address that is different from the number already there, you see the screen change at that location. What you see on the screen is an ASCII or graphic representation of the number, not the number itself, because when the number in the memory location is transferred to the screen, it is converted to the ASCII or graphic character it represents. If you put the number 79 in a video memory location you would not see 79 on the screen. To put the number 79 on the screen, you would have to issue two POKE commands because the character 7 takes one print location (memory address) and 9 takes another. Think of 79 as two different ASCII code characters, not as a number.

When dealing with the video display, think of the POKE statement as a faster PRINT@xxxxx,CHR$(ccc) command. The command POKE-15360,191 is like the command PRINT@0,CHR$(191)—each will put a solid graphic block at the upper left-hand corner of the video display. For example, we could paint the screen white with this short line:

```
9 FOR I = 15360 TO 16383:POKEI,191:NEXT I
```

Although POKExxxxx,ccc and PRINT@ xxxxx,CHR$(ccc) are alike, POKE has two advantages. First, it is slightly faster, and second, a POKE statement only affects one location on the screen, no matter where it is on the screen. The cursor always follows a PRINT statement, but a POKE command does not affect it. The PRINT command sometimes generates a line feed (even if you use a semicolon), whether you want it to or not; POKE does not. To get an idea of what this means, try to PRINT a solid graphic block (or any character) on the bottom right hand corner of the screen and have it stay there. Since 1023 is the PRINT@ location for that spot, press the CLEAR key, and type in this command:

```
PRINT@1023, CHR$(191);
```

It prints in the corner, but an automatic line feed moves it up one line. Even if you add on to the same statement the command:

```
:PRINT@ 0,"";
```

to move the cursor back up to the top of the screen, it still would not work. Press the CLEAR key to move the cursor to the top of the screen (to avoid a line feed caused when you press ENTER, which might cause the screen display to move up) and then type in POKE16383,191 (16383 is the last location of video memory). This time the graphic block stays where it is supposed to stay. POKE is the only way to put any character in this spot and have it stay there. Program Listing 3 illustrates a solution to this problem. If the last character of the last string were not POKEed, but PRINTed, the whole screen display would have moved up one line, and the top line of the screen display would have been lost.

PEEK Explained

PEEK looks, or peeks, at a certain address to see what is there. Just as POKE puts a certain number into a memory location, PEEK can look at what number is in a memory location. PEEK can look at any memory location, even if it's in ROM (read only memory). It works with the keyboard, and you can see what key or keys are being pressed. It does not change that address in any way; therefore, it is safe to use PEEK at any address to see what is there. If xxxxx is the address you want to look at, the PEEK command is written as PEEK(xxxxx). The statement P=PEEK(15360) will look at the top left part of the video display (memory address 15360) and assign to P the decimal value of the number it saw when it PEEKed. For example, if we printed the words MY GAME starting at the top left of the display, the letter M would be at the top left of the screen, and since 15360 is the address there, P=PEEK(15360) will return with the number 77, the ASCII code number for the letter M. Now we know that M is at the top left-hand corner of the display. If we wanted to know what was just to the right of it, we could PEEK at address 15361 (one place to the right) and see what was there. In this case P would return with the number 89. The command PRINT CHR$(89) will return a Y. See Program Listings 3 and 4 for examples of the PEEK function. In effect, we could look at the whole screen with this short line:

```
9 FOR I = 15360 TO 16383:P = PEEK(I):NEXT I
```

The POINT command for SET and RESET is similar to PEEK, but it tells you only if a particular pixel is on or off, not what letters or other graphic characters are in that print location. You can see what is in any print location (all six pixels) with the PEEK command. Now you can see if a car or ship in your program crashed or was hit by a missile from your program by PEEKing at the location where it should be. If a space ship is represented by a solid graphic block, CHR$(191), and the missiles by a single graphic pixel like CHR$(130), PEEKing at the location will tell you if a missile is there by returning the number 130 or if the ship is still there by returning the number 191. If your ship is larger and takes up more than one memory location, you can still check, but now you will have to PEEK at more locations.

Converting PRINT and POKE

Although the POKE command is fast, it is still practical to use the PRINT commands in some instances, so it is good to know how to convert from one to the other. To find out what address to POKE on the screen from a PRINT@xxxx,CHR$(ccc) command, simply add 15360 to the number xxxx. The number 15360 is also the first video memory address. To convert the statement PRINT@ 64,CHR$(191) to an equivalent POKE statement, add 15360 to 64. 15360 + 64 = 15424, so POKE 15424,191 accomplishes the same thing. Remember that POKExxxxx,ccc and PRINTCHR$(ccc) each use the same ASCII or graphic code number (ccc). Each statement will print a graphic block at the beginning of the second line of the video display. The reverse is possible, also. To find out where to PRINT@ from a POKE statement, subtract 15360 from the POKE address. The statement POKE16320,191 is converted this way: 16320 − 15360 = 960, so PRINT@960,CHR$(191) is equivalent to the POKE command. Each will put a graphic block on the bottom left-hand corner of the video display.

SET and RESET

Sometimes the SET and RESET commands are too slow. If you wanted to use the SET command to generate graphics that would look like one PRINT CHR$(191) or POKExxxx,191 (a solid graphic block), it would take six steps. The SET command can only SET one pixel of a print location at a time. Since there can be up to six pixels used at any one print location or address, it would take up to six SET commands to accomplish this.

The SET command is good for drawing lines at an angle, circles, and designs using formulas, but if you need things like this printed continuously, it is better to convert the result to numbers (location and character code) that you can POKE into a program line or simply to POKE and PRINT CHR$(xx) commands. This is especially true when more than one section of a print location is turned on at the same time. You can also save the video as strings once the display is set up, so the screen can be quickly printed again (see Program Listing 5). In either case, the PEEK command can help you do this.

The SET (X,Y) command uses two numbers: the x- and the y- axis numbers. Each SET command turns on one pixel of a print location (video memory location) at a time. Any combination of pixels in a print location can be turned on, but a SET command has to be used for each one. The graphic codes used by PRINT and POKE can accomplish any of the possible combinations with just one number. For example, to SET the top two pixels of a print location, two SET commands are needed, but only one POKE or PRINT CHR$(ccc) command is needed. The CHR$ and POKE code number for this example is 131.

Converting SET and RESET

To convert a SET(X,Y) command to a PRINT or POKE command, we have to know two things: where the print location is and what combination of pixels are on, or SET, at that location. To find out where it is, you use this formula: PRINT@ location = INT (X/2) + (INT (Y/3)*64).

X has the range of zero through 127, for a total of 128. 128/2 = 64. You divide by two because each print location has a total of two pixels across (two across by three down). The number of print locations per line is 64 as discussed earlier. Y has the range of zero through 47, for a total of 48. 48/3 = 16. You divide by three because each print location has a total of three pixels down (two across by three down). You multiply by 64 to get the exact position on the line. 16 is the number of lines of the video display; so the print location on the line (X/2) plus the line number times 64 (Y/3 times 64) gives the exact PRINT@ location. To convert this to a POKE or PEEK address, you add 15360.

Multiplying 128 by 48 (total x- by total y-axis) gives you the number 6144. This is the total number of pixels that can be turned on by SET commands. This is the same number you arrived at when you multiplied (64 × 16) × 6 = 6144. 64 × 16 is the total number of print locations on the screen, and six is the total number of pixels in each print location. This illustrates how the different commands are related.

Once you have found out where the print location is you have to find out what combination and how many of the six possible pixels are on, or SET. All you have to do is look, or PEEK, at the address you obtained.

For example, if part of your program has two SET commands: SET (120,2) and SET(120,1), the address of the first SET command is INT (120/2) + (INT(2/3)*64) = 60. INT (120/2) = 60;INT (2/3) = 0.0 × 64 = 0; and 60 + 0 = 60, so the PRINT location is 60. The PEEK an POKE address is 60 + 15360 = 15420. The address of the next SET command is INT (120/2) + (INT (1/2)*64). INT (120/2) = 60, and INT (1/2) = 0, so 15420 is the address of the second SET command. Both addresses are the same. To find out what is there, you look at the screen from your program by issuing this command P=PEEK(15420). P returns with the value of 148, so if you PRINT@60,CHR$(148) or POKE15420,148 you will accomplish the same thing with one command that took two SET commands. This is an example of how POKE and PEEK can help you convert programs to work faster and accomplish the same thing with fewer commands.

The Programs

The two main programs are Screen Saver (Program Listing 3) and Video Map (Program Listing 4). Each of these programs uses the statement: IF P < 32 THEN P = P + 64. This is needed for computers with lowercase modifications that do not have the driver program active. Sometimes when a PEEK command is used, an uppercase letter causes a number less than 32 to return as the ASCII code for that character. This would spell disaster if it were put into a string, because numbers under 32 are control codes for line feeds, back spaces, and the like. Adding 64 to these numbers will cause them to have the correct ASCII code.

Screen Saver

The Screen Saver program (Program Listing 3) shows how to save the video display in the form of strings. It can be used to save an important display, usually graphic, that has to be displayed rapidly and often. A formula for some SET commands might be used to draw what is needed, with the result saved as strings. There is one string for each line of the video. You can POKE these strings into program lines, or just use them in the program. You can modify the program so it will save only a part of the screen if that is all you need to save. Make sure that the cursor is where you want the strings to be printed before you print them, or use PRINT@ statements.

The program is used to illustrate how PEEK and POKE can help to control the video. PEEK looks at the screen to save it, and POKE helps put it back on the display. The POKE statement is needed to avoid a line feed caused by a PRINT statement when a character is printed at the last location of the video display. This is why the last string, S$(16), was treated separately. The first 63 characters were printed to avoid printing the 64th character in the last location. The last character was then converted to its ASCII equivalent so it could be POKEd on the screen—avoiding the line feed.

Video Map

Video Map (Program Listing 4) converts what is seen on the screen to what the computer sees. It is also good for showing where certain address or print locations are on the screen. To use the program, position the cursor (the blinking solid graphic block) over whatever character you want to analyze. Pressing the space bar once will cause the program to print the address, print location, code number of the character, and the character on the top line of the video display. If the cursor is on the top line of the screen, and you press the space bar again, you will get information about the information. Press the arrow keys to restore the original line. Press the CLEAR key to return to the original program. The program will help you understand how the video screen is organized and show you a practical application of the PEEK and POKE function when used for keyboard and video control. It can be used to help in the conversion from one type of video display to another by showing you where everything is and what the computer is seeing.

You can easily add some lines to store the information you want to remember in a variable array by adding Program Listing 5. H5 is a counter for the variable. Press the ENTER key to save the information about where the cursor is. If you want to save more than 200 locations in the array, change the DIM J(200,2) statement accordingly. The information can be put into data statements in your program to be read and POKEd back onto the screen. This is only practical when a small number of locations is being checked. If you want to save the whole screen display as strings, you can use Screen Saver (Program Listing 3).

The POKE function can be used to put information on the screen but should not be considered the best way to accomplish this. Using POKE in conjunction with the PRINT, and even the SET commands, will result in a well-rounded and more efficient program. We covered the use of POKE as far as the video display was concerned, but it can be used for many other functions; for example, you can POKE in a machine-language routine or even a BASIC program. Remember that POKE is a very powerful and dangerous command, because you might put something in a reserved area of memory used in the control of the computer.

The PEEK command is a safer but just as powerful function that lets you look at any memory location desired. Every BASIC word has a code number, which the computer uses to save a program, so you can analyze the program by the code numbers you see when you PEEK. You can use this function to look at, among other things, the video display and the keyboard.

Program Listing 1. *Keyboard control key check*

Encyclopedia Loader™

```
 9 :
   '  ARROW, SPACE, CLEAR, ENTER, AND BREAK KEY ADDRESS CHECK
19 :
   '
29 CLEAR 100 :
   C = 14400 :
   :
   '  C IS ADDRESS OF KEYS
39 CLS :
   PRINT TAB(10)"* PRESS ANY COMBINATION * "
49 P = PEEK(C) :
   :
   '  FIND OUT WHAT NUMBER IS THERE.
59 :
   '              0 MEANS NO KEYS ARE BEING PRESSED.
69 PRINT @ 448,"NUMBER IN ADDRESS 14400 :";
79 PRINT @ 476,"      "; :
   :
   '   ERASE LAST NUMBER PRINTED.
89 PRINT @ 476,P; :
   :
   '  PRINT NEW NUMBER.
99 GOTO 49 :
   :
   '  CHECK ADDRESS AGAIN.
```

Program Listing 2. *ASCII and graphic code display*

```
  9 CLS :
    PRINT TAB(10)" * PRINT LOCATION TEST *"
 19 PRINT " PRESS <SPACE> FOR NEXT CHARACTER "
 29 PRINT " HOLD DOWN <SPACE> FOR NO PAUSE "
 39 FOR I = 1 TO 255 :
     :
     ' LOOP FOR ASCII CHARACTER CODE NUMBER.
 49  PRINT @ 192," CHARACTER CODE NUMBER :";
 59  PRINT @ 216,"    "; :
     :
     '  ERASE LAST NUMBER.
 69  PRINT @ 216,I; :
     :
     ' PRINT NUMBER.
 79  POKE 16000,191:
     :
     '       POKE PRINT LOCATION WITH FULL GRAPHICS
 89  POKE 16000,I :
     :
     '      POKE PRINT LOCATION WITH NUMBER I.
 99  P = PEEK(14400) :
     :
     '       CHECK KEYBOARD FOR SPACE BAR.
119  IF P < > 128
      THEN
       GOTO 79 :
       :
       '  NO SPACE BAR DEPRESSED.
129  NEXT I
139 :
    '
149 :
    '  NOTE : IF YOU HAVE A LOWERCASE MODIFICATION YOU
159 :
```

Program continued

```
       '         MIGHT SEE MORE LETTERS IN RANGE 1-32.
169 :
       ' GRAPHICS ARE IN RANGE 128 TO 191.
```

Program Listing 3. *Screen Saver*

```
9899 :
     '  SCREEN SAVER :  A PROGRAM TO SAVE SCREEN AS STRINGS.
9912 :
     '
9919 CLEAR 2000 :
     DIM S$(16) :
     :
     '        ONE STRING FOR EACH LINE.
9929 FOR I = 15360 TO 16320 STEP 64 :
      :
      '  START AT THE BEGINNING
9931  :
      '                              ADDRESS OF EACH LINE.
9936  CC = CC + 1 :
      :
      '               SET UP COUNTER FOR STRINGS 1 - 16.
9939  FOR L = I TO I + 63 :
       :
       '      LOOK AT EACH ADDRESS OF LINE.
9949   P = PEEK(L) :
       :
       '      P= ASCII OR GRAPHIC CODE OF ADDRESS.
9952   IF P < 32
        THEN
         P = P + 64 :
         :
         '  LOWERCASE MODIFICATION NEEDS THIS.
9959   S$(CC) = S$(CC) + CHR$(P) :
       :
       '  ASSIGN CODE TO STRING.
9969   POKE L,32 :
       :
       '         ERASE PRINT LOCATION AFTER SAVING.
9979   NEXT L :
      NEXT I :
     :
     '    CONTINUE LOOPS.
9989 INPUT " PRESS ENTER TO SEE SCREEN ";X :
     CLS
9991 :
     '  PRINT SCREEN.
9992 FOR I = 1 TO 15 :
      PRINT S$(I); :
      NEXT I :
     :
     '  PRINT SCREEN.
9995 PRINT LEFT$(S$(16),63);:
     :
     ' DON'T PRINT LAST CHARACTER.
9996 P = ASC( RIGHT$(S$(16),1)):
     :
     ' GET ASCII CODE OF LAST CHARACTER.
9997 POKE 16383,P :
     :
     '  POKE LAST CHARACTER OF LAST STRING TO
9998 :
     '               AVOID LINE FEED THAT PRINT WOULD CAUSE.
9999 FOR X = 1 TO 999 :
      NEXT X :
     :
     '  DELAY A WHILE.
```

Program Listing 4. *Video Map*

```
9000 :
     '  VIDEO MAP.
9002 :
     '
9010 :
     '  BY, HUNT K. BRAND
9012 :
     '  3531 SAN CASTLE BLVD
9014 :
     '  LANTANA, FLORIDA. 33462
9015 :
     '  (305) 586-2377
9020 :
     '
9030 :
     '  USE A GOSUB 9000 TO ACTIVATE.
9035 :
     '  NOTE : ISSUE A CLEAR 1000 STATEMENT BEFORE USING
9038 :
     '         THIS PROGRAM.
9040 DEFINT A - Z
9050 C = 14400 :
     :
     '    ADDRESS OF ARROWS, CLEAR AND SPACE KEY.
9060 LA = 15360 :
     :
     '    ADDRESS OF LOW VIDEO MEMORY.
9070 HA = 16383 :
     :
     '    ADDRESS OF HIGH VIDEO MEMORY.
9080 CC = 191 :
     :
     '    CURSOR CHARACTER.
9090 C1 = CC :
     :
     '    CURSOR CHARACTER.
9100 LC = 15840 :
     :
     '    START CURSOR LOCATION.
9120 B$ = "" :
     FOR I = LA TO LA + 63 :
      P = PEEK(I)
9125  IF P < 32
       THEN
        P = P + 64
9130  B$ = B$ + CHR$(P) :
      POKE I,143 :
      NEXT I :
     :
     '  ASSIGN B$ VALUE OF EACH ADDRESS.
9140 PRINT @26," VIDEO MAP "; :
     H1 = 5
9145 :
     '      ---- INSTRUCTIONS ----
9150 :
     '  USE ARROW KEYS TO MOVE CURSOR.
9160 :
     '  USE SPACE BAR TO PRINT INFORMATION ON SCREEN.
9170 :
     '  USE CLEAR KEY TO RETURN TO MAIN PROGRAM.
9180 :
     '  TO PRINT INFO ALL THE TIME USE SPACE BAR WITH ARROWS.
9190 P = PEEK(C) :
     IF P = 0
      THEN
       9300
9200 IF P > 128
      THEN
```

Program continued

```
       P = P - 128 :
       H = 5 :
      ELSE
       H = 0
9210 IF P < > 128 AND H = 0 AND H1 = 5
      THEN
       PRINT @0,B$; :
       H1 = 0
9220 IF P = 8
      THEN
       LC = LC - 64 :
       IF LC < LA
        THEN
         LC = LC + 64 :
         :
         '  MOVE UP.
9230 IF P = 16
      THEN
       LC = LC + 64 :
       IF LC > HA
        THEN
         LC = LC - 64 :
         :
         '  MOVE DOWN.
9240 IF P = 64
      THEN
       LC = LC + 1 :
       IF LC > HA
        THEN
         LC = LC - 1 :
         :
         '  MOVE RIGHT.
9250 IF P = 32
      THEN
       LC = LC - 1 :
       IF LC < LA
        THEN
         LC = LC + 1 :
         :
         '  MOVE LEFT.
9260 IF P = 2
      THEN
       PRINT @0,B$; :
       RETURN :
       :
       '           ' CLEAR KEY.
9270 IF (P < > 128 AND H = 0)
      THEN
       9300 :
       :
       '     PRINT SCREEN INFORMATION.
9280 PRINT @0,"ADD =";LC;"   --   PRINT @ =";LC - LA;
9290 PRINT "   --"; TAB( POS(0) + 4)"CODE=";P1; TAB(55);
9295 PRINT "CHR$ = "; TAB(63) CHR$(P1); :
     H1 = 5
9300 P1 = PEEK(LC) :
     :
     ' CHARACTER AT CURSOR POSITION.
9310 IF P1 < 32
      THEN
       P1 = P1 + 64 :
       :
       ' LOWERCASE MODIFICATION
9311 :
     '                          WITHOUT DRIVER NEEDS THIS.
9320 IF P1 = 191
      THEN
       C1 = 32 :
      ELSE
       C1 = CC :
       :
```

```
       '  MAKE CURSOR DIFFERENT.
9330 POKE LC,C1
9340 FOR X = 1 TO 40 :
      IF PEEK(C) = 0
       THEN
        NEXT X
9350 POKE LC,C1 :
     POKE LC,P1
9360 FOR X = 1 TO 40 :
      P = PEEK(C) :
      IF P = 0
       THEN
        NEXT X :
       GOTO 9300 :
      ELSE
       9200
```

Program Listing 5. *Additions for Video Map*

```
9045 DIM J(200,2) :
     :
     ' ARRAY TO SAVE SCREEN INFORMATION
9046 :
     '                FOR VIDEO MAP.
9048 :
     '  J(A,2) WHERE J(A,1) IS VIDEO ADDRESS.
9049 :
     '               J(A,2) IS CHARACTER AT THAT ADDRESS.
9251 IF P = 1
      THEN
       H5 = H5 + 1 :
       :
       ' ENTER KEY WAS PRESSED.
9252 :
     '  H5 IS THE COUNTER FOR THE VARIABLE
9253 IF P = 1
      THEN
       J(H5,1) = LC :
       J(H5,2) = P1 :
       :
       '  SAVE INFO.
```

TUTORIAL

Hairy Bi-Nary and Hexy-Decimal

by Joe D. Fugate

I doubt if any subject scares the average TRS-80 user more quickly than the mention of binary or hexadecimal—it's almost like some kind of electronic virus. So, if binary is not a type of terminal disease and hexadecimal is not some kind of curse—just *what* are they and *how* do they work?

Binary

Bi means two, like a bicycle has two wheels. Ary means pertaining to. Binary is pertaining to two, referring to two numbers, 0 and 1. Why those two numbers? That's because your computer knows only two states: off and on. Off is the same as 0, on is the same as 1. Your computer can tell if a particular place in memory is either charged up (1) or turned off (0), and it starts counting all those ones and zeros in a special math called binary math.

Binary math, first lesson: Each one or zero is called a digit, or more exactly a binary digit, called a bit for short.

Binary math, second lesson: Let's learn to count in binary. Zero, one—now what? You can't use two, because that's not a bit (binary digit) value. Only zero and one can be used. So what comes after one? It has to be the next larger combination of 1 and 0. So what's next? 2? No. 3? No. 4? No. 5? No. 6? No. (I'm being repetitive to point out that none of these are binary digits.) 7? No. 8? No. 9? No. 10? YES! That 10 looks like a ten, but it's not. We counted to one in binary and were looking for two, but we couldn't use that, because we had to stick with zeros and ones, and we came to 10. So using zeros and ones only, 10 becomes the binary number for two. It follows that 11 is three. Binary is extremely cumbersome, but remember, all the machine can understand is zeros (off) and ones (on). So here's what we have:

```
0 = 0    1 bit (Binary digIT)
1 = 1
2 = 10   2 bits
3 = 11
4 = 100   3 bits
5 = 101
6 = 110
7 = 111
8 = 1000   4 bits
```

Now let's do the counting over, always using four bits in computer memory:

```
0 = 0000
```

$$\begin{array}{l}
1 = 0001\\
2 = 0010\\
3 = 0011\\
4 = 0100\\
5 = 0101\\
6 = 0110\\
7 = 0111\\
8 = 1000
\end{array}$$

Let's take those four bits and give each bit position a value, based on its position, like this:

$$\begin{array}{rcll}
0 & = & 0\ 0\ 0\ 0 & \\
 & & 8\ 4\ 2\ 1 & \text{value}
\end{array}$$

Notice that each value is *two* times the following value, because it's binary, and bi means two. Let's say that if we have a 1 above that value, we'll use the value to figure the decimal equivalent of the binary number. Here's an example:

$$\begin{array}{rcl}
2 & = & 0\ 0\ 1\ 0\\
 & & 8\ 4\ \underline{2}\ 1
\end{array}$$

It should be two, and it is! How about another try:

$$\begin{array}{rcll}
5 & = & 0\ 1\ 0\ 1 & \text{Sum of values:}\\
 & & 8\ \underline{4}\ 2\ \underline{1} & 4 + 1 = 5
\end{array}$$

Again, it works. Here are some more examples:

$$\begin{array}{rcll}
8 & = & 1\ 0\ 0\ 0 & \\
 & & \underline{8}\ 4\ 2\ 1 & 8 = 8\\
10 & = & 1\ 0\ 1\ 0 & \\
 & & \underline{8}\ 4\ \underline{2}\ 1 & 8 + 2 = 10\\
15 & = & 1\ 1\ 1\ 1 & \\
 & & \underline{8\ 4\ 2\ 1} & 8 + 4 + 2 + 1 = 15
\end{array}$$

It looks like fifteen is the maximum number we can store here. What if we try eight bits instead of four? Remember, we just multiply each new position by two to get its value. That gives us:

$$\begin{array}{cccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
128 & 64 & 32 & 16 & 8 & 4 & 2 & 1
\end{array}$$

The maximum number we can store in eight bits is:

$$\begin{array}{cccccccc}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1\\
\underline{128} & \underline{64} & \underline{32} & \underline{16\ 8\ 4\ 2\ 1} & & & & 128+64+32+16+8+4+2+1 = 255
\end{array}$$

Because eight bits is a nice even number of bits to work with, the block of eight bits has a name; it's called a byte. Therefore, the biggest number you can store in one byte is 255.

Hexadecimal

Hex means six. A hexagon has six sides. Decimal means ten. Hexadecimal literally means six-ten or more exactly, 16.

Let's digress for a moment. It takes eight bits to equal one byte, so in binary, let's write out four bytes of all zeros:

```
00000000  00000000  00000000  00000000
```

That's 32 bits (eight bits each, times four bytes). It would be nice if we didn't have to write out all those bits. There is a way—hexadecimal.

The key to understanding hexadecimal is to realize that it's nothing more than shorthand binary. Let's see how that works. First off, in order to make each byte more manageable, let's break its eight bits into two groups of four bits, like this:

```
0000  0000
0000  0000
  4     4
  1 byte
```

Now let's take each four-bit group and come up with a single character code to represent every possible four-bit combination. That's easy for zero through nine:

```
0000 = 0
0001 = 1
0010 = 2
0011 = 3
0100 = 4
0101 = 5
0110 = 6
0111 = 7
1000 = 8
1001 = 9
```

Notice that the code up to this point is the same as the actual numerical value of the bits.

From the discussion on binary, we know that the largest number we can store in four bits is 15 (1111). In our code assignment above, we stopped at nine. What about 10 through 15, now that we are out of single-character numbers. Let's just use the first six letters of the alphabet for the remaining codes, like this:

```
1010 = A (ten)
1011 = B (eleven)
1100 = C (twelve)
1101 = D (thirteen)
1110 = E (fourteen)
1111 = F (fifteen)
```

Let's see how hexadecimal numbers work in actual practice. Since we have divided our byte into two groups of four bits, we can now use two hexadecimal code characters to represent a byte:

Example 1

$$\begin{array}{cc} \underline{0000} & \underline{0000} \\ 0 & 0 \\ \uparrow & \uparrow \end{array}$$

Hexadecimal
codes

Hex representation is 00
Actual numeric value = 0

Example 2

$$\begin{array}{cc} \underline{0001} & \underline{1000} \\ 1 & 8 \\ \uparrow & \uparrow \end{array}$$

Hexadecimal
codes

Hex representation is 18
Actual numeric value = 16 + 8 = 24

Example 3

$$\begin{array}{cc} \underline{1111} & \underline{0111} \\ F & 7 \end{array}$$

Hex representation is F7
Actual value = 128 + 64 + 32 + 16 + 4 + 2 + 1 = 247

Example 4

$$\begin{array}{cc} \underline{1100} & \underline{1010} \\ C & A \end{array}$$

Hex representation is CA
Actual value = 128 + 64 + 8 + 2 = 202

It's a lot easier to write F7 than 1111 0111. Notice that one byte will always be represented by two hex digit codes.

Here is a problem for you to figure out yourself: What is the actual numeric value of FF? (Hint: Remember hex is shorthand binary.) Did you get 255? First convert FF back to binary. F equals 1111, so FF is 1111 1111. List out the place values for each bit:

1	1	1	1	1	1	1	1	bits
128	64	32	16	8	4	2	1	place value

Now let's add up all of the one-bit place values, which in this case is all of the place values. The sum is 255, so hex FF equals 255.

Now when you see things like "END OF RAM = 4FFF" you won't be intimidated.

Instant Indexer: Programming in Disk BASIC

by Del Gomes, John Jewell and Alan Zendner

Would you like a neat, alphabetically listed menu on each disk, so that all you'd have to do is type in the number of the program that you'd like? Does having a list of your programs sorted by disk, program name, or both sound even better? If the answer is yes, then this program design should be of interest to you. With Instant Indexer, all these features can be generated by your TRS-80 by merely entering and running this program.

The programs assist your microcomputer in creating a disk directory for each disk using NEWDOS. They'll also allow you to merge the directories from each disk with varied output, and provide a distinctive menu selection for each disk. For those who are beginning Disk BASIC or who have done limited programming in Disk BASIC, things do get easier. You have a whole new world in front of you: easy access, quick loading, and new instructions (TRSDOS, NEWDOS+, NEWDOS/80, and VTOS). If you are a little overwhelmed by the newness of it all, then the techniques employed here may be of even greater interest than the actual programs. Their chief value is in the ideas and disk programming concepts that they help you develop.

Video Memory

The key to using your Disk BASIC and Disk Operating System to create the disk directory is in reading video memory. There is a specific area of RAM devoted to what appears on your monitor screen. Each of the Radio Shack manuals contains the addresses (from 15360 to 16383). Once you realize the possibilities that evolve from being able to input data from the computer display, you may want to write your own routine. (Then see if we could have done a better job.)

If you have never examined your video memory before, let's try a few simple exercises. First get into BASIC, then press the CLEAR button, and type a row of As across the top of the screen. You'll receive a syntax-error message when you press ENTER. Now type in PRINT PEEK(15360). Your TRS-80 responds with 65. Type in PRINT CHR$(65) and it responds A. Aha! Now try PRINT CHR$(PEEK(15360)), the result is A. Press

CLEAR again and type in your name. You're now ready to run the following program:

```
 10  CLS
 20  LINEINPUT"MY NAME IS ";N$
 30  CLS
 40  PRINT N$
 50  PRINT:PRINT
 60  L = LEN(N$)
 70  FOR I = 0 TO L - 1
 80  X = PEEK(15360 + I)
100  PRINT CHR$(X);
110  NEXT
120  END
```

By this time, you should have a feel for being able to get data off the screen and using it in some practical fashion. Oh no! Someone out there said that this did not work for him at all. If it didn't work for you, and you have a lowercase modification, relax. All you have to do is to run through an additional step or two. This doesn't mean a change in technique, but in execution. As you probably know, for every simple rule that you learn for the TRS-80, there are at least three exceptions. With a lowercase modification, PRINT CHR$(PEEK(15360)) just makes a blank line. That's because Tandy uses the lower decimal codes to handle part of their alphabet. However, if you type in PRINT CHR$(PEEK(15360) + 64), the result should be an A. Note that because you have both lower- and uppercase, you must check the program to see if PEEK(15360) < 32. If it is, add 64, as follows:

```
90 IF X < 32 THEN X = X + 64
```

Reading the Directory

This very nice feature of capturing information from the screen in combination with NEWDOS + ability to issue DOS commands using CMD, simplifies this program. First the program calls FREE to determine: the name of the disk, the date it was created, the number of files, and grans remaining. The standard entry reads:

```
DRIVE 0 — DISKNAME   MM/DD/YY   23 FILES   5 GRANS
```

If you have more than one drive, this information is repeated for each drive. The data for Drive 0 will always be on the first line, for Drive 1 on the second, and so forth. Taking this information from the screen requires PEEKing at the appropriate video-memory locations. Line 1 is contained in RAM from 15360 to 15423; line 2 from 15424 to 15487. Since the drive number isn't needed, we begin PEEKing at 15372 to get the name/number for the disk on Drive 0 (which this program is for). This data is recorded in DISK$.

The next step is to find out what programs and files are on the disk. This only works for those programs and files that appear when you type DIR upon entering DOS. The disk directory will not list invisible or system files. The search is done by issuing the CMD DIR:X (where X is the desired disk drive). Any programs or files found are displayed beginning on the third line and continuing up to the fourteenth line. When you have 36 or more programs on a disk, the DIR command requires you to hit the ENTER key to see all the names. This places a limit of 35 on the number of program entries this routine can handle. For most people at the beginning level, this should present no problem.

To find out how many lines there are, you can quickly PEEK at the first character in each row. As soon as the routine returns a code 32, you know that line is empty.

```
200  FOR Z = 0 TO 14
210  Y = PEEK(15488 + Z*64)
220  IF Y = 32 THEN LAST = Z - 1
230  NEXT Z
```

This simple check saves PEEKing line after line of blank screen.

There are a number of ways of retrieving program names from video memory. As so frequently happens in programming, we tried several in developing our final routine. The first procedure discussed is not the most efficient, nor the shortest, but, it handles the information in single steps which are easily discussed.

For each line J of video, PEEK at the contents of the entire line. While there are other methods technically equivalent, such as PEEKing in blocks of 254, reading line-by-line gives a clear picture of what this portion of the program is doing. Each line on the screen is read into a string array (A$(J)) running from one to LAST. When the process is finished, the string array contains a mirror image of your monitor screen.

Finally, to get the data from the video memory, each program name must be isolated from the one to three names each line string contains. Each line in your DIR listing contains a maximum of three evenly-spaced entries. Since each line has 64 spaces, it is logical that the program names begin at 20 space intervals: 1, 21, and 41. (If you want to check this fact, just use CMD DIR from BASIC and the PEEK routines discussed earlier.) This program works in combination with the BASIC functions MID$ and INSTR. Indeed, MID$ is specifically designed to return a substring given its beginning position (B) and length (L). (If we omit the length, then

MID$ will return all of the right-hand portion of the string from the starting position to the end.) Let's look at a sample string in our array:

```
A$(J) = "PROGRAM1      PROGRAM2      PROGRAM3"
```

Then B$(K) = MID$(A$(J), B1, L1)—where B1 is the initial position and L1 the length of the name—will return PROGRAM1. Similarly B$(K + 1) = MID$(A$(J),B2,L2) will return PROGRAM2. B$(K + 2) = MID$(A$(J), B3,L3) will provide PROGRAM3.

You already know that B1 = 1, B2 = 21, and B3 = 41. All that you need now is the length of the program names. Ah, but their length can vary from one to 12 characters. INSTR to the rescue! INSTR returns the initial position of a substring in the original string. This formula:

```
E1 = INSTR( B1, A$(J), " ")
```

specifies where the blank portion following the first name in the string begins. Note that L1, the length of PROGRAM1, is equal to E1–B1. Similarly, for L2 and L3:

```
E2 = INSTR( B2, A$(J), " ")
E3 = INSTR( B3, A$(J), " ")
```

Because this function indicates the position in the main string to begin the search, it is a very powerful and handy tool. Inserting these values into our MID$ routine effectively separates out the desired names.

These steps may be repeated as many times as you need, until the LAST line is read and analyzed. Using two loops to execute the string analyses discussed makes this process shorter and more efficient. In the final line, a check is made to insure that none of the strings, B(K + 1)$ or B(K + 2)$, is empty. Of course, if the second string is null, then so is the third. One simple way of determining this is to see if the value of E2 or E3 returned by INSTR is equal to the starting position, B2 or B3. If they are equal, the element is empty.

Reading the Efficient Way

Our final DISKDIR routine is a simpler, more direct method than the one outlined above. Rather than approaching the video memory line-by-line, we read each program name individually. The first step is to determine the number of lines containing programs. Be aware that you could integrate this into the second step by checking to see if the first character of each name is blank. The line-check method is kept so that the emphasis is solely upon examining the name.

Step two develops a series of operations that point the variable X to the beginning of each program name. The FOR loop with ROW controls the line or row placement; the FOR loop with COL determines the column. An I counter determines the element of a string array:

```
240  FOR ROW = 0 TO LAST
250  W = 15488 + ROW*64
260  FOR COL = 0 TO 2
270  I = I + 1
280  X = W + COL*20
```

With X in the correct position, you can read the name character-by-character directly into your string array. An A$(I) serves as a temporary storage for each character. These characters are added one at a time into B$(I) which serves as the final holder. The end of the name has been reached when you receive a PEEKed code of 32. The check for the lower-case modification is made by looking for codes less than 32.

```
290  A = PEEK(X)
300  IF A = 32 THEN 350
310  IF A < 32 THEN A = A + 64
320  A$(I) = CHR$(A)
330  B$(I) = B$(I) + A$(I)
340  X = X + 1 : GOTO 290
```

These steps from 290 to 340 cycle until the name has been read in its entirety. At this point, X is at the blank and A equals 32. The routine jumps to line 350 to start next column:

```
350  NEXT COL
```

The result is a return to lines 260 and 270 where the I counter increases and points to the next element in the string array, and is set to the second column in the line. After another cycle, X is set to the third column. When

```
360  NEXT ROW
```

is reached, the process begins again for a new line.

It is possible that one or two of the final strings are empty. The initial search for the number of lines merely guarantees that the first position of the final line is occupied. Therefore, a quick check is made on the status of the strings:

```
370  M = I
380  FOR L = M TO M-1 STEP -1
```

```
390  IF B$(L) = "" THEN I = I - 1
400  NEXT L
```

Each one of the program names listed in the disk's directory is contained in the string array B$(1) to B$(I).

Sorting the Array

If you are using disks, you probably have encountered numerous sort techniques, from bubble to Shell-Metzner to machine language. Any good string sort will suffice. Because most disk files are under forty programs, even the slow bubble sort would be adequate. For a faster method, use the very rapid sort provided by Allan Emert in the July 1980 issue of "TRS-80 Microcomputer News" and included here in Program Listings 3 and 4. Note that memory size must be set at 48895 for 32K and 65279 for 48K systems.

Add good utilities such as this one to your regular programs. They can save you considerable time in handling standard situations. Adding a sort is simple, if it is in BASIC. Type in the routine as you normally would, with the listing beginning at line 5000 or higher, but save it as an ASCII text:

```
SAVE "RAPIDSRT/TXT",A
```

When a sort would be just the answer to a problem, it may be quickly added by MERGEing. Load whatever program you have developed, making sure that none of the line numbers in your program coincides with those in RAPIDSRT/TXT. Then MERGE"RAPIDSRT/TXT". The lines in the sort routine have been sequentially inserted into your program. If any lines had coincided, your program lines would have been replaced by those of RAPIDSRT/TXT.

Subroutines or portions of other programs can be merged easily with any program. You can build an extensive, usable library of subprograms ready to be merged whenever needed.

Creating Disk Files

At this point, you have the file of all the program names on disk sorted into alphabetical order. For those of you who are still learning the elements of Disk BASIC, let's review how to create a file of these names. We'll be using sequential files. With sequential files, you sacrifice the ability to retrieve a single record out of order. But since the names are already in order, and the whole list is printed, sequential files are appropriate.

possible in all forms of NEWDOS. TRSDOS users are able to issue the ATTRIB command within BASIC only with NEWDOS+, and its later edition, NEWDOS/80.

To record the information on the disk, we use PRINT#1, B$(J) where 1 is the number of the buffer specified as the opened file and B$(J) is an element in the sorted array. As when you have the computer print information on the screen, you must specify delimiters to separate the different components. Because you don't want the strings to run together, semicolons can't be used. Though the comma can be used, it is not an efficient way to utilize disk space. Putting a comma after a string causes the next string to be printed a full tab setting away. Adding a comma contained in quotes to the end of each element, except the last, provides a solution:

```
160  PRINT#1, DISK$
420  FOR J = 1 TO I - 1
430  PRINT#1, B$(J) + ", ";
440  NEXT J
450  PRINT#1, B$(I)
460  CLOSE
```

Congratulations! You've created a sequential disk file with the names of all the programs you've placed on that disk. The hard work is over. It's time to have some fun using your new DISKDATA file.

Disk Menu

The menu program has three parts: opening and reading your new DISKDATA file; printing the name of the disk and each of the program listings; and running your selection. Opening a sequential file for input from disk resembles the steps followed for sequential output:

```
60  OPEN "I", 1, "DISKDATA"
```

An I specifies sequential input to buffer 1 from your directory file. To transfer the information from disk into the TRS-80 memory banks, we use INPUT# and LINEINPUT#. The first is suitable for the regular string array. The second is necessary for DISK$, which contains punctuation marks, the disk name, and the number of files and grans.

Printing this information in the desired format is simple, straightforward programming. Read in disk data, and print the name and information about files/grans on the top line:

```
70  LINEINPUT#1, DISK$
80  PRINT "FILE DIRECTORY: " + DISK$
```

Then in three columns, and as many rows as necessary, print a number and corresponding program name. There are a number of ways to do this. The following method is not the fastest, but it mirrors the way in which the information was obtained. A FOR loop with ROW controls the line placement; A FOR loop with COL determines the column location:

```
90   FOR ROW = 1 to 14
100  FOR COL = 0 TO 2
```

Now you're ready to read in the program names. An I counter marks the elements in the string array again. When the PRINT TAB(COL*20) command places the cursor in the proper column, the element number and name are printed:

```
110  I = I + 1
120  INPUT#1, B$(I)
130  PRINT TAB (COL*20)
140  PRINT USING F$; I;
150  PRINT B$(I);
```

After each program name has been read and printed, the routine checks to see if that element is the last in the sequential file. If not, then the next column is engaged:

```
160  IF EOF(1) THEN 200
170  NEXT COL
```

When all three program names have been printed, drop to the following line and call the next ROW:

```
180  PRINT CHR$(13);
190  NEXT ROW
200  CLOSE
```

This last step of closing the file is critical; always be sure that it is done. The routine will work without properly closing DISKDATA. The trouble comes later. Killing an open file can result in ruining all the information stored on that disk.

The final part of the disk menu is constructing a routine to run the program chosen. The usual procedure is to use an INPUT or LINEINPUT to allow the user to designate his choice. NEWDOS+ makes it easy to execute CMD programs from BASIC:

```
220  LINEINPUT"NUMBER SELECTED: "; S$
```

```
230  L = VAL(S$)
240  IF RIGHT$(B$(L),4) = "/CMD" THEN CMD B$(L) ELSE RUN B$(L)
```

Our personal preference in menu routines is for INKEY$. But, an attractive user-oriented alternative is easily constructed:

```
220  PRINT "PROGRAM SELECTION IS: ";
230  S$ = INKEY$
240  IF S$ = "" THEN 230
250  IF (VAL(S$) > 9 OR VAL(S$) < 1) THEN 230
260  PRINT S$;
```

In these lines, S$ is first checked to see if any key has been pressed. If some entry was made, line 250 ensures that only the numbers one through nine can be input.

These are the regular steps to obtain a single input from INKEY$. If a disk has more than nine programs, a second INKEY$ is added. That is simple. Allowing for choice of a program designated by a single digit is the tricky part. The solution is a timing loop:

```
270  T$ = INKEY$
280  IF C = 80 THEN 330
290  C = C + 1
300  IF T$  "" THEN 270
310  IF (VAL(T$) > 9 OR VAL(T$) < 1) THEN 270
320  PRINT T$
330  S = VAL(S$ + T$)
```

The steps from lines 270 to 290 will cycle for 80 counts. This is long enough to enter a second number if you want. Experiment. Try changing the value of C until the routine is right for you and your users. If it doesn't work well for you, then you can always use the regular ENTER and INPUT process.

The success of this brief menu depends upon the care with which you name your programs. If you have a machine-language program and omit the /CMD after the name, your TRS-80 has no way of knowing how to respond. Though abbreviations may appear useful in saving keystrokes, typing in the full program name provides an attractive screen display and it helps you remember just what each program is. After that, all you have to do is enter one or two numbers with an INKEY$ routine.

Applications

An early application for the DISKDIR was to solve the problem of trying to remember what disk the programs were on. The DISKDATA can create a master file for program libraries. This master file can supply the following: a listing of all the programs in our disk library, a disk-by-disk inventory, an accounting of which disk grans are still available, and a file on each disk which can be loaded into Scripsit for those who don't have NEWDOS/80. This is convenient when you are in Scripsit but can't remember just what you named the file you want to load. It is also very simple. DISKDATA is an ASCII file, so load it as you would any text created on Scripsit.

Program Listing 1. *Disk directory program, 32K*

```
  0 '        ***********************************************
  1 '        *               DISK DIRECTORY PROGRAM        *
  2 '        * --BY Del Gomes, John Jewell & Alan Zendner-- *
  3 '        ***********************************************
  4 '
  5 '
                 >>>>  INITIALIZATION OF PROGRAM  <<<<
 10 CLEAR 8000
 15 :
    '        ====  Disk Entry Point For 32K Machine Language Sort  =
    ===
 20 DEF USR = &HBF00
 25 :
    '        ====  Define & Dimension Variables  ====
 30 DIM A$(100), B$(100), X(2)
 40 DEFINT A - Z :
    :
    '  LAST, ROW, COL ARE INTEGER VARIABLES.
 45 :
    '        ====  Poke Sort Routine  ====
 50 GOSUB 5090
 55 :
    '                ==== Open Sequential File  ====
 60 OPEN "O",1,"DISKDATA:0"
 70 CMD "ATTRIB DISKDATA:0 (I)"
 80 CLS
 85 :
    '                  >>>>  DETERMINE DISK NAME & NO. OF GRAMS  <<<<

 90 CMD "FREE"
100 FOR X = 15372 TO 15423
110  A = PEEK(X)
120  IF A < 32
      THEN
       A = A + 64
130  D$ = CHR$(A)
140  DISK$ = DISK$ + D$
150  NEXT X
160 PRINT #1, DISK$
170 CLS
175 :
    '                  >>>>  CREATE ARRAY OF PROGRAM NAMES  <<<<

180 CMD "DIR :0"
190 I = 0
195 :
    '        ====  Determine Lines Containing Programs  ====
200 FOR Z = 0 TO 14
210  Y = PEEK(15488 + Z * 64)
220  IF Y = 32
      THEN
       LAST = Z - 1 :
       GOTO 240
230  NEXT Z
235 :
    '        ====  Peek Program Names Into String Array ====
240 FOR ROW = 0 TO LAST
250  W = 15488 + ROW * 64
260  FOR COL = 0 TO 2
270   I = I + 1
280   X = W + COL * 20
290   A = PEEK(X)
300   IF A = 32
       THEN
        350
310   IF A < 32
       THEN
```

```
        A = A + 64
320   A$(I) = CHR$(A)
330   B$(I) = B$(I) + A$(I)
340   X = X + 1 :
      GOTO 290
350   NEXT COL
360  NEXT ROW
365 :
    '          ====  Eliminate Empty Strings From Last Line  ====
370 M = I
380 FOR L = M TO M - 1 STEP - 1
390  IF B$(L) = ""
      THEN
       I = I - 1
400  NEXT L
405 :
    '          ====  Sort Program Names Into Alphabetical Order  ====

410 GOSUB 5050
415 :
    '                  >>>>  WRITE PROGRAM NAMES TO DISK  <<<<
420 FOR J = 1 TO I - 1
430  PRINT #1, B$(J) + ", ";
440  NEXT J
450 PRINT #1, B$(I)
455 :
    '                  >>>>  CLOSING PORTION OF PROGRAM  <<<<
460 CLOSE
470 CLS
480 PRINT CHR$(23)
490 PRINT @ 256, "DISKDATA FILE HAS BEEN CREATED."
500 FOR K = 1 TO 1500
510  NEXT K
520 CLS
530 END
5000 :
     '            **************************************************
5010 :
     '              MACHINE-LANGUAGE SORT FOR BASIC PROGRAMS
                      Memory Size = 48895
5020 :
     '          Source:  TRS-80 Microcomputer News, July 1980,
                  pp.1+
5030 :
     '          Program Modified For 32K System With Disk Drive.
5040 :
     '         **************************************************

5050 X(0) = I
5060 X(1) = VARPTR(B$(1))
5070 Z = USR( VARPTR(X(0)))
5080 RETURN
5090 DATA 205,127,10,94,35,86,237,83,19,191,35,94,35,86,237,83
5100 DATA 213,191,33,0,0,34,211,191,237,91,211,191,203,59,175
5110 DATA 203,58,48,2,203,251,237,83,211,191,122,179,200,42,19
5120 DATA 191,237,82,34,207,191,33,0,0,34,205,191,42,205,191,34
5130 DATA 203,191,42,203,191,237,91,211,191,25,34,209,191,235,33
5140 DATA 0,0,25,25,25,229,237,91,203,191,33,0,0,25,25,25,237
5150 DATA 75,213,191,9,235,225,9,229,213,14,0,126,71,26,184,48
5160 DATA 3,14,1,71,175,176,40,25,197,19,35,78,35,70,197,225
5170 DATA 235,78,35,70,197,225,193,26,150,56,10,32,39,19,35,16
5180 DATA 246,203,65,32,31,209,225,6,3,78,235,126,113,235,119
5190 DATA 35,19,16,246,42,211,191,235,42,203,191,175,237,82,34
5200 DATA 203,191,48,144,24,2,209,225,42,205,191,17,1,0,175,25
5210 DATA 34,205,191,237,91,207,191,237,82,218,58,191,195,24,191
5220 N = 0
5230 FOR I = 1 TO 203
5240  READ A
5250  N = N + A
5260  POKE I - 16641,A
```

Program continued

```
5270  NEXT
5280 IF N < > 23865
      THEN
       END
5290 RETURN
```

Program Listing 2. *Disk directory program, 48K*

```
  0 '         ************************************************
  1 '         *                 DISKMENU                     *
  2 '         * --BY Del Gomes, John Jewell & Alan Zendner-- *
  3 '         ************************************************
  4 '
  5 :
    '                  >>>>  INITIALIZATION OF PROGRAM  <<<<
 10 CLEAR 8000
 15 :
    '         ====  Disk Entry Point For 48K Machine Language Sort  =
    ===
 20 DEF USR = &HFF00
 25 :
    '         ====  Define & Dimension Variables  ====
 30 DIM A$(100), B$(100), X(2)
 40 DEFINT A - Z :
    :
    '  LAST, ROW, COL ARE INTEGER VARIABLES.
 45 :
    '         ====  Poke Sort Routine  ====
 50 GOSUB 5090
 55 :
    '               - ====  Open Sequential File  ====
 60 OPEN "O",1,"DISKDATA:0"
 70 CMD "ATTRIB DISKDATA:0 (I)"
 80 CLS
 85 :
    '                  >>>>  DETERMINE DISK NAME & NO. OF GRAMS  <<<<

 90 CMD "FREE"
100 FOR X = 15372 TO 15423
110  A = PEEK(X)
120  IF A < 32
      THEN
       A = A + 64
130  D$ = CHR$(A)
140  DISK$ = DISK$ + D$
150  NEXT X
160 PRINT #1, DISK$
170 CLS
175 :
    '                  >>>>  CREATE ARRAY OF PROGRAM NAMES  <<<<

180 CMD "DIR :0"
190 I = 0
195 :
    '         ====  Determine Lines Containing Programs  ====
200 FOR Z = 0 TO 14
210  Y = PEEK(15488 + Z * 64)
220  IF Y = 32
      THEN
       LAST = Z - 1 :
       GOTO 240
230  NEXT Z
235 :
    '         ====  Peek Program Names Into String Array ====
240 FOR ROW = 0 TO LAST
250  W = 15488 + ROW * 64
```

```
 260  FOR COL = 0 TO 2
 270   I = I + 1
 280   X = W + COL * 20
 290   A = PEEK(X)
 300   IF A = 32
        THEN
         350
 310   IF A < 32
        THEN
         A = A + 64
 320   A$(I) = CHR$(A)
 330   B$(I) = B$(I) + A$(I)
 340   X = X + 1 :
       GOTO 290
 350   NEXT COL
 360  NEXT ROW
 365 :
     '          ====  Eliminate Empty Strings From Last Line  ====
 370 M = I
 380 FOR L = M TO M - 1 STEP - 1
 390  IF B$(L) = ""
       THEN
        I = I - 1
 400  NEXT L
 405 :
     '          ====  Sort Program Names Into Alphabetical Order  ====
 410 GOSUB 5050
 415 :
     '                   >>>>  WRITE PROGRAM NAMES TO DISK  <<<<
 420 FOR J = 1 TO I - 1
 430  PRINT #1, B$(J) + ", ";
 440  NEXT J
 450 PRINT #1, B$(I)
 455 :
     '                   >>>>  CLOSING PORTION OF PROGRAM  <<<<
 460 CLOSE
 470 CLS
 480 PRINT CHR$(23)
 490 PRINT @ 256, "DISKDATA FILE HAS BEEN CREATED."
 500 FOR K = 1 TO 1500
 510  NEXT K
 520 CLS
 530 END
5000 :
     '               *************************************************
5010 :
     '              MACHINE-LANGUAGE SORT FOR BASIC PROGRAMS
                       Memory Size = 65279
5020 :
     '          Source:  TRS-80 Microcomputer News, July 1980,
                     pp.1+
5030 :
     '          Program Modified For 48K System With Disk Drive.
5040 :
     '         *************************************************
5050 X(0) = I
5060 X(1) = VARPTR(B$(1))
5070 Z = USR( VARPTR(X(0)))
5080 RETURN
5090 DATA 205,127,10,94,35,86,237,83,19,255,35,94,35,86,237,83
5100 DATA 213,255,33,0,0,34,211,255,237,91,211,255,203,59,175
5110 DATA 203,58,48,2,203,251,237,83,211,255,122,179,200,42,19
5120 DATA 255,237,82,34,207,255,33,0,0,34,205,255,42,205,255,34
5130 DATA 203,255,42,203,255,237,91,211,255,25,34,209,255,235,33
5140 DATA 0,0,25,25,25,229,237,91,203,255,33,0,0,25,25,25,237
5150 DATA 75,213,255,9,235,225,9,229,213,14,0,126,71,26,184,48
5160 DATA 3,14,1,71,175,176,40,25,197,19,35,78,35,70,197,225
5170 DATA 235,78,35,70,197,225,193,26,150,56,10,32,39,19,35,16
5180 DATA 246,203,65,32,31,209,225,6,3,78,235,126,113,235,119
```

Program continued

```
5190 DATA 35,19,16,246,42,211,255,235,42,203,255,175,237,82,34
5200 DATA 203,255,48,144,24,2,209,225,42,205,255,17,1,0,175,25
5210 DATA 34,205,255,237,91,207,255,237,82,218,58,255,195,24,255
5220 N = 0
5230 FOR I = 1 TO 203
5240  READ A
5250  N = N + A
5260  POKE I - 257,A
5270  NEXT
5280 IF N < > 25337
      THEN
       END
5290 RETURN
```

Program Listing 3. *Machine-language sort for BASIC programs, 32K*

```
5000 '
5005 '       **************************************************
5010 '            MACHINE-LANGUAGE SORT FOR BASIC PROGRAMS
5015 '                       Memory Size = 48895
5020 '        Source:  TRS-80 Microcomputer News, July 1980,
5025 '              pp.1+
5030 '        Program Modified For 32K System With Disk Drive.
5040 '       **************************************************
5045 '
5050 X(0) = I
5060 X(1) = VARPTR(B$(1))
5070 Z = USR( VARPTR(X(0)))
5080 RETURN
5090 DATA 205,127,10,94,35,86,237,83,19,191,35,94,35,86,237,83
5100 DATA 213,191,33,0,0,34,211,191,237,91,211,191,203,59,175
5110 DATA 203,58,48,2,203,251,237,83,211,191,122,179,200,42,19
5120 DATA 191,237,82,34,207,191,33,0,0,34,205,191,42,205,191,34
5130 DATA 203,191,42,203,191,237,91,211,191,25,34,209,191,235,33
5140 DATA 0,0,25,25,25,229,237,91,203,191,33,0,0,25,25,25,237
5150 DATA 75,213,191,9,235,225,9,229,213,14,0,126,71,26,184,48
5160 DATA 3,14,1,71,175,176,40,25,197,19,35,78,35,70,197,225
5170 DATA 235,78,35,70,197,225,193,26,150,56,10,32,39,19,35,16
5180 DATA 246,203,65,32,31,209,225,6,3,78,235,126,113,235,119
5190 DATA 35,19,16,246,42,211,191,235,42,203,191,175,237,82,34
5200 DATA 203,191,48,144,24,2,209,225,42,205,191,17,1,0,175,25
5210 DATA 34,205,191,237,91,207,191,237,82,218,58,191,195,24,191
5220 N = 0
5230 FOR I = 1 TO 203
5240  READ A
5250  N = N + A
5260  POKE I - 16641,A
5270  NEXT
5280 IF N < > 23865
      THEN
       END
5290 RETURN
```

Program Listing 4. *Machine-language sort for BASIC program, 48K*

```
5000 '
5005 '       **************************************************
5010 '            MACHINE-LANGUAGE SORT FOR BASIC PROGRAMS
5015 '                       Memory Size = 65279
5020 '        Source:  TRS-80 Microcomputer News, July 1980,
5025              PP.1+
5030 '       Program Modified For 48K System With Disk Drive.
5040 '      ************************************************
5045 '
5050 X(0) = I
5060 X(1) = VARPTR(B$(1))
5070 Z = USR( VARPTR(X(0)))
5080 RETURN
5090 DATA 205,127,10,94,35,86,237,83,19,255,35,94,35,86,237,83
5100 DATA 213,255,33,0,0,34,211,255,237,91,211,255,203,59,175
5110 DATA 203,58,48,2,203,251,237,83,211,255,122,179,200,42,19
5120 DATA 255,237,82,34,207,255,33,0,0,34,205,255,42,205,255,34
5130 DATA 203,255,42,203,255,237,91,211,255,25,34,209,255,235,33
5140 DATA 0,0,25,25,25,229,237,91,203,255,33,0,0,25,25,25,237
5150 DATA 75,213,255,9,235,225,9,229,213,14,0,126,71,26,184,48
5160 DATA 3,14,1,71,175,176,40,25,197,19,35,78,35,70,197,225
5170 DATA 235,78,35,70,197,225,193,26,150,56,10,32,39,19,35,16
5180 DATA 246,203,65,32,31,209,225,6,3,78,235,126,113,235,119
5190 DATA 35,19,16,246,42,211,255,235,42,203,255,175,237,82,34
5200 DATA 203,255,48,144,24,2,209,225,42,205,255,17,1,0,175,25
5210 DATA 34,205,255,237,91,207,255,237,82,218,58,255,195,24,255
5220 N = 0
5230 FOR I = 1 TO 203
5240  READ A
5250  N = N + A
5260  POKE I - 257,A
5270  NEXT
5280 IF N < > 25337
      THEN
       END
5290 RETURN
```

Program Listing 5. *Diskmenu*

```
  0 '        ************************************************
  1 '        *              DISK DIRECTORY PROGRAM          *
  2 '        * --BY Del Gomes, John Jewell & Alan Zendner-- *
  3 '        ************************************************
  4 '
  5 '
                      >>>>  INITIALIZATION OF PROGRAM  <<<<
 10 CLEAR 2000
 15 :
    '
                         >>>> DEFINE & DIMENSION VARIABLES <<<<
 20 DIM B$(100)
 30 DEFINT A - Z
 40 F$ = "## " :
    I = 0 :
    C = 0
 45 :
    '
                         >>>> READ & WRITE DISKDATA <<<<
 50 CLS
 60 OPEN "I", 1, "DISKDATA:0"
 70 LINE INPUT #1, DISK$
 80 PRINT "FILE DIRECTORY    " + DISK$
 90 FOR ROW = 1 TO 14
100  FOR COL = 0 TO 2
110   I = I + 1
120   INPUT #1, B$(I)
130   PRINT TAB(COL * 20)
140   PRINT USING F$; I;
150   PRINT B$(I);
160   IF EOF (1)
       THEN
        200
170   NEXT COL
180  PRINT CHR$(13);
```

Program continued

```
190  NEXT ROW
200 CLOSE
210 PRINT :
    PRINT
215 :
    '                  >>>> SELECT & EXECUTE PROGRAM <<<<
220 PRINT "PROGRAM SELECTION IS: ";
230 S$ = INKEY$
240 IF S$ = ""
     THEN
      230
250 IF ( VAL(S$) > 9 OR VAL(S$) < 1)
     THEN
      230
260 PRINT S$;
270 T$ = INKEY$
280 IF C = 80
     THEN
      330
290 C = C + 1
300 IF T$ = ""
     THEN
      270
310 IF ( VAL(T$) > 9 OR VAL(T$) < 1)
     THEN
      270
320 PRINT T$
330 S = VAL(S$ + T$)
340 CLS
350 PRINT @ 394, "LOADING FILENAME: "; B$(S)
360 IF RIGHT$(B$(S),4) = "/CMD"
     THEN
      CMD B$(S) :
     ELSE
      RUN B$(S)
370 END
```

UTILITY

Uni-Key for the Model I
BREAK Disable
Z-80 Disassembler

Uni-Key for the Model I

by Rowland Archer Jr.

One night while I was trying to massage some life into my tired fingers after a couple of hours at the keyboard, it occurred to me that typing programs is the sort of drudgery a computer is supposed to take out of life, not put into it. Typing a BASIC program repeats many of the same BASIC keywords over and over.

To pass some of this work off on the TRS-80, I needed a way to let it know which keyword I wanted with a single-key abbreviation. Using computerized keywords would also cut down on syntax errors and tedious editing.

Lowercase ASCII Code

Even though BASIC is uppercase only on the TRS-80, the keyboard will generate lowercase ASCII character codes. Try this short BASIC program and see what happens:

```
10 CLS
20 A$ = ""
30 A$ = INKEY$: IF A$ = "" THEN GOTO 30
40 PRINT@0,ASC(A$);: GOTO 20
```

Run the program and press any alphabetic key, say A. The ASCII code for uppercase A, 65, should appear on the screen. Now press SHIFT A; the code printed should be 97, which is lowercase a.

To get lowercase letters on a TRS-80, you press the SHIFT key. Although it may seem backwards to shift for lowercase, would you rather have to shift for uppercase? You would have to hold down the SHIFT key to enter every BASIC keyword. A SHIFT-lock key would get around this problem, but apparently Radio Shack didn't feel the need for one.

A routine which examines every pressed key before the character value is returned to the BASIC interpreter program is necessary. When the TRS-80 appears to be doing nothing, it is actually reading the keyboard over and over, waiting for a key to be pressed. Assuming it is possible to intercept characters from the keyboard and look them over before they reach BASIC, it is possible to decide whether to send the character on to BASIC as is, send back some other character instead, or even send BASIC a stream of two or more characters in its place.

To accomplish this, it is necessary to install a filter between the keyboard and the BASIC interpreter. This would be a device whose action filters input data streams to produce output data. We need a filter which translates some input characters to BASIC keywords and leaves others alone. If the character we intercept from the keyboard is an uppercase letter, a number, or a special symbol (@, ?, +, etc.), our program should pass the character on to the caller unchanged.

If it is a lowercase letter (a-z), however, which is transmitted when the user hits the SHIFT key and a letter, the filter should replace that letter in the input stream with a BASIC keyword. To do this, when a lowercase letter is read from the keyboard, the program sets a substitution flag in the routine, indicating that keyword substitution has begun. Use the ASCII value of the letter as an index to a table of BASIC keywords. Pass BASIC the first letter of the indexed keyword instead of the lowercase letter read from the keyboard.

When BASIC calls for input from the keyboard again, the routine will note that the substitution flag is set and will send back the next character of the keyword, without bothering to look at the keyboard to see if any keys are pressed. This continues until the entire keyword is sent. Then the flag is reset to normal operation until the next lowercase letter code is received. All this happens so quickly that the keyword seems to appear on the screen the instant the key is pressed.

Note that only 26 keywords can be handled with this method. It is possible, however, to select a group of keywords which are either frequently used, difficult to type, or both. You can experiment by including different keywords until you find the best subset for your needs. I have found the set of 26 shown in Table 1 to be very useful.

Intercepting Input

Since the routine will be used while editing BASIC programs, the easiest approach is to write an assembly-language routine to look at characters before the BASIC editor scans them. Take a look at the memory map in the back of your Level II BASIC manual. At location 4015H (hexadecimal) in the BASIC reserved RAM area, there is a device control block for the keyboard.

Location 4016H is initialized by BASIC with the address of the keyboard driver routine. A call to this routine returns with 0 in the A register if no key is pressed, or the ASCII value of the key if one is pressed (like an assembly-language INKEY$ routine).

Load the routine into high memory where it can be protected from BASIC by answering the MEMORY SIZE? prompt appropriately on startup. When it is first run, it grabs the address of the keyboard driver routine from location 4016H and saves it for later use. It then stores the address of its own entry point at 4016H so that every routine (in ROM or elsewhere) which used to call the keyboard driver directly will now call this routine instead.

The normal keyboard driver is called as a subroutine to read from the keyboard. If no key is pressed, the subroutine will return a zero to the accumulator (register A); in this case we return to the caller without changing a thing. If a key has been pressed, it is examined as discussed above, and if it is a lowercase alphabetic code, the keyword substitution routine begins.

A User-Defined Key

The above technique allows entry of 26 BASIC keywords, each with a single keystroke, and by itself will save a lot of typing. But many BASIC programs use the same expression over and over; it would be convenient to enter a phrase once and then recall it with a single keystroke. To do this, a key must be defined whose substitution value can be changed dynamically, without having to reassemble and reload the assembly-language routine. This feature is easily added by declaring one key as the "define user string" key. This routine uses SHIFT/CLEAR. The "substitute user defined string" key is the shifted down arrow.

These keys can be located in the input stream in the same way shifted alphabetic characters are intercepted. When SHIFT/CLEAR is pressed, a START DEFINITION prompt is printed on the screen. Each character typed, up to 64 characters, is saved in memory until SHIFT/CLEAR is hit again. This terminates the definition of the string. END DEFINITION is written on the screen. Now when SHIFT plus the down arrow are pressed, the defined string is returned. Table 2 gives a summary of the actions the filter routines will perform for the range of possible keyboard inputs.

The origin shown in the Program Listing of ONESTR is for a 16K machine. Table 3 gives the ORG value to substitute, as well as the appropriate answers to the MEMORY SIZE? prompt, for 32K and 48K machines. At label INIT you will find the initialization code, which must be executed once when ONESTR is loaded. This code retrieves the address of the current keyboard driver routine from the keyboard device control block and stores it after the CALL opcodes at labels KEYDR1 and KEYDR2.

Level II BASIC, TRSDOS 2.1, 2.2, and NEWDOS all use different keyboard drivers. By picking up the address they have already put in the device control block, it is possible to reap the benefits (debounce, etc.) of these drivers and the benefits of ONESTR at the same time. The address of label ONESTR is installed in the device control block so that it is called in the future for keyboard input.

Choose an Instruction

Choose one of two instructions, depending on whether you will be loading ONESTR from tape or disk. If you will be loading ONESTR from disk, you must use JP 402DH to return to TRSDOS or NEWDOS. If you will be

SHIFT for Keywords:

A	PRINT@	N	NEXT
B	ELSE	O	POKE
C	CHR$(	P	PEEK(
D	DATA	Q	LEFT$(
E	RIGHT$(	R	RETURN
F	FOR	S	GOSUB
G	GOTO	T	TAB(
H	RND(	U	USING
I	INPUT"	V	STRING$(
J	READ	W	MID$(
K	INKEY$	X	SET(
L	LEN(	Y	THEN
M	ASC(	Z	RESET(

CLEAR—START/END definition
Down arrow—user string

Table 1. *Reference chart of keywords*

loading from tape with the SYSTEM command in BASIC, use JP 0072H to return to BASIC. Note that the space taken up by this initialization code can be reused after it is run, since the code is no longer needed. The answers to the MEMORY SIZE? prompt are given in Table 3 and reflect this reuse of space.

At label USTR allocate 64 bytes for the user-defined string and give it an initial value of RUN. Until it is redefined, typing SHIFT/down arrow will

Input	ASCII Value	Output
Special characters	0–25	Same as input
SHIFT/down arrow	26	Substitute user-defined string (up to 64 characters)
Special characters	27–30	Same as input
SHIFT/CLEAR	31 & Shift*	None (start/end user string definition)
Special characters and numbers	32–64	Same as input
Uppercase letters	65–90	Same as input
Special characters	91–96	Same as input
Lowercase letters	97–122	Substitute a BASIC keyword
Special characters	123–127	Same as input

*Since the keyboard driver returns 31 for both SHIFT/CLEAR and CLEAR, we must test for the shift key separately.

Table 2. *Function of keyboard filter routine*

run a BASIC program. This may be changed to any other initial value you like, and the code at label REST will allocate the remainder of the 64-byte buffer. A zero byte after the string serves as the string terminator.

Beginning at label LA the table of strings to be substituted for the lowercase letters are listed. Except for LA, the labels are included strictly for convenience in determining which keyword gets substituted for which letter; they are not needed by the code. This is where you would substitute assembly-language mnemonics, Pascal keywords, or anything else you would like to type with one keystroke. If you do make substitutions which cause the length of the program to change, be sure to change the program origin so it will fit in your machine. Adjust your answer to the MEMORY SIZE? prompt in this case to protect the new size of the program. Again, a zero byte at the end of each string in the table serves as a terminator.

RAM Size	ORG of Program	Answer MEMORY SIZE?
16K	7E58H	32361
32K	BE58H	48745
48K	FE58H	65129

*Since the keyboard driver returns 31 for both SHIFT/CLEAR and CLEAR, we must test for the shift key separately.

Table 3. *Origin of program and MEMORY SIZE for different RAM sizes*

Selecting Keywords

I chose not to include an entry for PRINT, which is certainly a commonly used BASIC keyword. The Level II BASIC handbook explains that a question mark is a built-in abbreviation for PRINT. I did choose to include PRINT@ as a keyword, even though typing ?@ works as well and is almost as easy. The reasoning behind this is that @ and SHIFT@ appear the same on the screen, but SHIFT@ doesn't work as a PRINT qualifier. It's a nasty bug to catch since a listing appears normal. Including PRINT@ as a one-stroke entry avoids the problem.

ONESTR is the main entry point to the program, and this is where control is transferred whenever keyboard entry is requested. The OS FLAG is tested to see if the routine is in the middle of a keyword substitution. If so, it branches to SUBST and continues with the substitution. Otherwise, the routine whose address was in the keyboard device control block before ONESTR was loaded is called for keyboard input. If no key has been pressed, it returns to the caller. If a key has been pressed, a decision is made about the next action to take, based on the key's ASCII value (as shown in Table 2).

If the keyboard input is a lowercase letter, the OS flag is set on. Search the keyword table sequentially to find the start of the keyword to substitute. The number of keywords to skip is calculated from the ASCII value of the lowercase letter read from the keyboard, minus the ASCII value of lowercase a. Since each keyword is terminated by a zero byte, start at the head of the table and check each character. Continue until you pass as many zero bytes as the number of keywords you are supposed to skip.

Don't moan and groan about the inefficiency of a sequential search—it's easy to build an index for the address table and to put it into the routine so that the appropriate address may be computed. To do this, use a DEFW pseudo-op instruction for each label whose address is included in the following table:

```
ADTABL  DEFW   LA
        DEFW   LB
        . . .
        DEFW   LZ
```

The offset into the table for a lowercase letter, say b, is 2* (ASCII(b)-ASCII(a)). The factor of 2 is used because of the two bytes each entry takes. Load the HL register pair with the contents of this location, and you have the address of the first letter of the keyword.

Why not use this technique, then, instead of the sequential search? Because the address table takes up space, and although it does give faster results, the difference isn't noticeable at the keyboard.

Test the Shift Key

The code at the label DEFINE is executed when the CLEAR key is pressed. Since the keyboard driver routine returns the same value whether CLEAR is pressed alone or shifted, a test must be done to verify that the SHIFT key is indicated. Location 3880H contains a 1 if SHIFT is pressed, and a 0 otherwise. Checking 3880H determines whether to start user string definition, which is triggered by SHIFT/CLEAR.

Assuming SHIFT/CLEAR was pressed, the program types DEFINE STRING: on the screen and waits for input. Each key pressed is tested to see if it is SHIFT/CLEAR, which ends the definition. If not, it is added to the user string buffer at USTR.

If 64 characters are typed without a SHIFT/CLEAR, then definition mode is automatically terminated. User string definition ends by returning a zero to the caller, indicating that no key was pressed. Thus the entire process is invisible to the caller. This process can be used with any program (BASIC, etc.) requesting keyboard input.

Note that most of the Level II BASIC string input editing is not implemented. The back arrow will delete a character, but SHIFT plus the back arrow will not delete the whole line. Another design trade-off is reflected here. The ROM routines for string input editing could have been used, but they terminate input when BREAK or ENTER are pressed. The approach taken allows entire commands to be typed with one keystroke, including ENTER at the end of the command.

Running ONESTR

If you are loading from disk, run ONESTR from the DOS READY prompt. You can test it at this point by typing shifted letters; keywords should appear. With either disk or tape, bring BASIC up, answering MEMORY SIZE? as shown in Table 3. Disk users should be in business at this point.

Tape users should enter the SYSTEM command and load the object tape. Once it is loaded, run the program by hitting ENTER.

Now it's time to find those back issues of *80 Microcomputing* and start enjoying all the programs that you were too lazy to type.

Since this program was written, TRSDOS 2.3 came out. It requires a little more help to keep it from clobbering this program when it loads BASIC from disk. After line 00270 in the program, add:

```
4049         00272  ORG    4049H    ; DISK USERS ONLY—PROTECT
4049  697E   00274  DEFW   USTR-1   ; ONESTR BY LOADING DOS'
                                    ; HIGH$ LOCATION WITH ADDRESS
                                    ; OF LAST BYTE OF USABLE
                                    ; MEMORY
```

This extra code is not needed but does no harm under NEWDOS. You can ignore the MEMORY SIZE? question (disk users only) if you add this code.

Program Listing. *ONESTR*

```
               ; ONESTR - ONE STROKE KEYWORD ENTRY PROGRAM.  INTERCEPTS
               ;        LOWER-CASE CHARACTERS AND REPLACES THEM WITH
               ;        KEYWORD STRINGS.  ALSO ALLOWS THE USER TO ASSIGN A
               ;        STRING OF UP TO 64 BYTES TO 'SHIFT-DOWN ARROW'.
               ;        DEFINITION OF THIS STRING IS INITIATED AND
               ;        TERMINATED BY 'SHIFT-CLEAR'.
               ;
               ;        REV 2.2         2/9/79
               ;
               ;        BY ROWLAND ARCHER
               ;           FLINT RIDGE 59
               ;           HILLSBOROUGH, NC  27278
               ;
4016           KEYDRV  EQU     4016H           ;ADDRESS IN KEYBOARD DCB
                                               ; OF DRIVER ROUTINE
3880           SHIFT   EQU     3880H           ;(3880H) IS 1 IF 'SHIFT'
                                               ; KEY IS PRESSED
033A           PUTC    EQU     033AH           ;PUT CHAR IN A ON SCREEN
00CD           CALL    EQU     0CDH            ;'CALL' OPCODE VALUE
001F           DEFKEY  EQU     31              ;'SHIFT-CLEAR' KEY
001A           UDSKEY  EQU     26              ;'RETURN USER-DEF STRING'
                                               ; KEY = 'SHIFT-DOWN ARROW'
0040           USTLEN  EQU     64              ;USER-DEFINED STRING LENGTH
               ;
               ;THE FOLLOWING INITIALIZATION CODE IS PERFORMED ONLY
               ;WHEN THIS ROUTINE IS LOADED AND RUN THE FIRST TIME.
               ;
7E58                   ORG     07E58H          ;ORG FOR 16K SYSTEM
7E58 2A1640    INIT    LD      HL,(KEYDRV)     ;GET ADDR OF KEYBRD DRIVER
7E5B 224E7F            LD      (KEYDR1),HL     ; ROUTINE AND BUILD TWO
7E5E 22B07F            LD      (KEYDR2),HL     ; CALL INSTRUCTIONS IN
                                               ; THIS CODE WITH IT
7E61 21477F            LD      HL,ONESTR       ;NOW PUT THE ADDRESS OF THE
7E64 221640            LD      (KEYDRV),HL     ; ENTRY PT TO THIS ROUTINE
                                               ; INTO THE KEYBOARD DCB
               ;*******************************************************
               ;YOU MUST CHOOSE ONLY ONE OF THE FOLLOWING TWO JUMP
               ;INSTRUCTIONS TO EXIT THIS INITIALIZATION CODE.
               ;IF YOU ARE GOING TO LOAD THIS PROGRAM FROM DISK WHILE
               ;IN DOS, USE
               ;       JP      402DH           ; TO RETURN TO DOS
               ;IF YOU ARE GOING TO LOAD FROM TAPE WHILE IN BASIC, USE
7E67 C3191A            JP      0072H           ; TO RETURN TO BASIC
               ;*******************************************************
               ;
               ; ABOVE CODE IS ONLY USED ONCE AND CAN BE OVERWRITTEN
               ; AFTER IT RUNS - SO 'MEMORY SIZE?' PROTECTIONS STARTS
               ; WITH THE FOLLOWING DATA STRUCTURES:
               ;
7E6A 52        USTR    DEFM    'RUN'           ;USER STRING; INITIALLY
7E6D 0D                DEFB    0DH             ; 'RUN <ENTER>'
7E6E 00                DEFB    0               ;END OF STRING
0005           USED    EQU     $-USTR          ;SIZE OF PREDEFINED STRING
003B           REST    DEFS    USTLEN-USED     ;ALLOCATE SPACE FOR REST
                                               ; OF USER-DEFINED STRING
7EAA 00                DEFB    0               ;FORCE END OF STRING
7EAB 00        OSFLAG  DEFB    0               ;ONE-STROKE FLAG: = 1 WHILE
                                               ; WE ARE SUBSTITUTING FOR A
                                               ; LOWER-CASE CHARACTER
7EAC 0000      OSPTR   DEFW    0               ;ADDRESS OF CURRENT CHAR
                                               ; IN SUBSTITUTE STRING
               ;
               ;TABLE OF STRINGS TO SUBSTITUTE FOR LOWER-CASE CHARS.
               ;STRINGS ARE TERMINATED BY NULL (0) BYTES.  STRING
               ;LABELLED 'LA' IS SUBSTITUTED FOR 'SHIFT-A', 'LB'
               ;FOR 'SHIFT-B', ETC.  EXCEPT FOR 'LA', LABELS
               ;ARE NOT REQUIRED, AND ARE ONLY INCLUDED FOR EASE IN
               ;DETERMINING THE STRING TO BE SUBSTITUTED FOR EACH LETTER.
               ;
7EAE 50        LA      DEFM    'PRINT@'
7EB4 00                DEFB    0               ;ZERO BYTE END OF STRING
7EB5 45        LB      DEFM    'ELSE'
7EB9 00                DEFB    0
7EBA 43        LC      DEFM    'CHR$('
7EBF ØØ         ØØ75Ø          DEFB     Ø
7ECØ 44         ØØ76Ø LD       DEFM     'DATA'
7EC4 ØØ         ØØ77Ø          DEFB     Ø
7EC5 52         ØØ78Ø LE       DEFM     'RIGHT$('
7ECC ØØ         ØØ79Ø          DEFB     Ø
7ECD 46         ØØ8ØØ LF       DEFM     'FOR'
7EDØ ØØ         ØØ81Ø          DEFB     Ø
7ED1 47         ØØ82Ø LG       DEFM     'GOTO'
7ED5 ØØ         ØØ83Ø          DEFB     Ø
7ED6 52         ØØ84Ø LH       DEFM     'RND('
7EDA ØØ         ØØ85Ø          DEFB     Ø
7EDB 49         ØØ86Ø LI       DEFM     'INPUT"'
7EE1 ØØ         ØØ87Ø          DEFB     Ø
7EE2 52         ØØ88Ø LJ       DEFM     'READ'
7EE6 ØØ         ØØ89Ø          DEFB     Ø
7EE7 49         ØØ9ØØ LK       DEFM     'INKEY$'
7EED ØØ         ØØ91Ø          DEFB     Ø
7EEE 4C         ØØ92Ø LL       DEFM     'LEN('
7EF2 ØØ         ØØ93Ø          DEFB     Ø
7EF3 41         ØØ94Ø LM       DEFM     'ASC('
7EF7 ØØ         ØØ95Ø          DEFB     Ø
7EF8 4E         ØØ96Ø LN       DEFM     'NEXT'
7EFC ØØ         ØØ97Ø          DEFB     Ø
7EFD 5Ø         ØØ98Ø LO       DEFM     'POKE'
7FØ1 ØØ         ØØ99Ø          DEFB     Ø
7FØ2 5Ø         Ø1ØØØ LP       DEFM     'PEEK('
7FØ7 ØØ         Ø1Ø1Ø          DEFB     Ø
7FØ8 4C         Ø1Ø2Ø LQ       DEFM     'LEFT$('
7FØE ØØ         Ø1Ø3Ø          DEFB     Ø
7FØF 52         Ø1Ø4Ø LR       DEFM     'RETURN'
7F15 ØØ         Ø1Ø5Ø          DEFB     Ø
7F16 47         Ø1Ø6Ø LS       DEFM     'GOSUB'
7F1B ØØ         Ø1Ø7Ø          DEFB     Ø
7F1C 54         Ø1Ø8Ø LT       DEFM     'TAB('
7F2Ø ØØ         Ø1Ø9Ø          DEFB     Ø
7F21 55         Ø11ØØ LU       DEFM     'USING'
7F26 ØØ         Ø111Ø          DEFB     Ø
7F27 53         Ø112Ø LV       DEFM     'STRING$('
7F2F ØØ         Ø113Ø          DEFB     Ø
7F3Ø 4D         Ø114Ø LW       DEFM     'MID$('
7F35 ØØ         Ø115Ø          DEFB     Ø
7F36 53         Ø116Ø LX       DEFM     'SET('
7F3A ØØ         Ø117Ø          DEFB     Ø
7F3B 54         Ø118Ø LY       DEFM     'THEN'
7F3F ØØ         Ø119Ø          DEFB     Ø
7F4Ø 52         Ø12ØØ LZ       DEFM     'RESET('
7F46 ØØ         Ø121Ø          DEFB     Ø
                Ø122Ø ;
                Ø123Ø ;MAIN ROUTINE ENTRY POINT:
                Ø124Ø ;
7F47 3AAB7E     Ø125Ø ONESTR   LD       A,(OSFLAG)      ;IF FLAG<>Ø WE ARE IN THE
7F4A B7         Ø126Ø          OR       A               ;MIDDLE OF A SUBSTITUTION
7F4B 2Ø38       Ø127Ø          JR       NZ,SUBST        ;CONTINUE SUBSTITUTION
                Ø128Ø ;
                Ø129Ø ;CALL NORMAL ROUTINE TO GET CHARACTER FROM KEYBOARD
                Ø13ØØ ;
7F4D CD         Ø131Ø          DEFB     CALL            ;BECOMES 'CALL GET-CHAR'
7F4E ØØØØ       Ø132Ø KEYDR1   DEFW     Ø               ; WHEN INITIALIZATION CODE
                Ø133Ø                                   ; PUTS ADDRESS OF KEYBOARD
                Ø134Ø                                   ; DRIVER ROUTINE HERE
7F5Ø B7         Ø135Ø          OR       A               ;CHARACTER RETURNED IN A
7F51 C8         Ø136Ø          RET      Z               ;Ø MEANS NO KEY PRESSED,
                Ø137Ø                                   ;SO JUST RETURN TO CALLER
                Ø138Ø ;
                Ø139Ø ;A KEY HAS BEEN PRESSED, HANDLE IT IF IT IS 'DEFKEY',
                Ø14ØØ ;'UDSKEY' OR LOWER-CASE LETTER; ELSE JUST RETURN IT.
                Ø141Ø ;
7F52 FE1F       Ø142Ø          CP       DEFKEY          ;DEFINE USER STRING?
7F54 284Ø       Ø143Ø          JR       Z,DEFINE        ;YES, GO DO IT
7F56 FE1A       Ø144Ø          CP       UDSKEY          ;REQUESTING USER STRING?
7F58 28Ø8       Ø145Ø          JR       Z,SUBMOD        ;YES, START SUBSTITUTION
7F5A FE61       Ø146Ø          CP       97              ;KEY < LOWER-CASE A?
7F5C D8         Ø147Ø          RET      C               ;YES, RETURN UNCHANGED
7F5D FE7B       Ø148Ø          CP       123             ;KEY < LOWER-CASE Z + 1?
7F5F 38Ø1       Ø149Ø          JR       C,SUBMOD        ;YES, SUBSTITUTE
7F61 C9         Ø15ØØ          RET                      ;NO, RETURN UNCHANGED
```

Program continued

```
             ;
             ;START NEW SUBSTITUTION - SET OSFLAG = 1,
             ;SET POINTER TO STRING TO SUBSTITUTE
             ;
7F62 E5      SUBMOD  PUSH    HL
7F63 FE1A            CP      UDSKEY          ;USER-DEFINED STRING?
7F65 2ØØ5            JR      NZ,KEYWRD       ;NO, IT'S KEYWORD
7F67 216A7E          LD      HL,USTR         ;YES, GO SAVE POINTER TO
7F6A 181Ø            JR      SRCHDN          ; STRING AND SET MODE FLAG
             ;
             ;KEY PRESSED WAS A LOWER-CASE LETTER.  SET POINTER TO FIRST
             ;CHARACTER OF KEYWORD TO SUBSTITUTE FOR IT.
             ;
7F6C 21AE7E  KEYWRD  LD      HL,LA           ;BASE OF SUBST-STRING TABLE
7F6F D661            SUB     97              ;SUBTRACT ASCII(LOWER-CASE
                                             ; A) FROM KEY PRESSED
7F71 B7              OR      A               ;ZERO => LOWER-CASE A
7F72 28Ø8            JR      Z,SRCHDN        ; PRESSED, END SEARCH
7F74 47              LD      B,A             ;ELSE A HOLDS NUMBER OF
7F75 7E      NXTC    LD      A,(HL)          ; KEYWORDS TO SKIP OVER TO
7F76 23              INC     HL              ; FIND STRING TO SUBSTITUTE
7F77 B7              OR      A
7F78 2ØFB            JR      NZ,NXTC         ;INNER LOOP FINDS NULL
                                             ; END-OF-STRING BYTES
7F7A 1ØF9            DJNZ    NXTC            ;OUTER LOOP COUNTS KEYWORDS
             ;
             ;END SEARCH - HL HAS POINTER TO DESIRED STRING
             ;
7F7C 22AC7E  SRCHDN  LD      (OSPTR),HL      ;SAVE POINTER TO STRING
7F7F 3EØ1    SETMD   LD      A,1             ;SUBSTITUTION MODE STARTS
7F81 32AB7E          LD      (OSFLAG),A
7F84 E1              POP     HL              ;RESTORE HL
             ;
             ;BRANCH HERE WHEN WE ARE DOING A SUBSTITUTION
             ;
7F85 E5      SUBST   PUSH    HL              ;SAVE HL
7F86 2AAC7E          LD      HL,(OSPTR)      ;GET CURRENT CHARACTER
7F89 7E              LD      A,(HL)          ; OF SUBSTITUTION STRING
7F8A B7              OR      A               ;NULL END-OF-STRING?
7F8B 2ØØ3            JR      NZ,NOTEND       ;NO, MORE TO GO
7F8D 32AB7E          LD      (OSFLAG),A      ;YES, END SUBSTITUTION
7F9Ø 23      NOTEND  INC     HL              ;BUMP POINTER TO NEXT
7F91 22AC7E          LD      (OSPTR),HL      ; CHARACTER AND SAVE IT
7F94 E1              POP     HL              ;RESTORE HL
7F95 C9              RET                     ;RETURN CHARACTER IN A
             ;
             ;DEFINITION OF USER STRING
             ;
7F96 3A8Ø38  DEFINE  LD      A,(SHIFT)       ;SHIFT DEPRESSED?
7F99 B7              OR      A               ; (DEFINE ON 'SHIFT-CLEAR')
7F9A 2ØØ4            JR      NZ,DEF          ;YES, DEFINE IT
7F9C 3E1F            LD      A,DEFKEY        ;NO, RETURN 'CLEAR'
7F9E B7              OR      A               ;SET FLAGS FOR CALLER
7F9F C9              RET                     ;RETURN CHAR IN A
             ;
7FAØ C5      DEF     PUSH    BC              ;SAVE CALLER'S BC
7FA1 E5              PUSH    HL              ;SAVE CALLER'S HL
7FA2 21DE7F          LD      HL,STRTDF       ;PUT PROMPT FOR START OF
7FA5 CDD57F          CALL    PUTSTR          ; USER STRING DEFINITION
7FA8 216A7E          LD      HL,USTR         ;POINTER TO USER STRING AREA
7FAB Ø64Ø            LD      B,USTLEN        ;MAX SIZE OF USER STRING
7FAD E5      GETC    PUSH    HL              ;SAVE OUR HL
7FAE C5              PUSH    BC              ;SAVE OUR BC
7FAF CD              DEFB    CALL            ;BECOMES 'CALL GET-CHAR'
7FBØ ØØØØ    KEYDR2  DEFW    Ø               ;ADDR OF KEY DRIVER HERE
7FB2 C1              POP     BC              ;RESTORE OUR BC
7FB3 E1              POP     HL              ;RESTORE OUR HL
7FB4 B7              OR      A               ;IS A NON-ZERO?
7FB5 28F6            JR      Z,GETC          ;LOOP UNTIL KEY PRESSED
7FB7 FE1F            CP      DEFKEY          ;END DEFINITION?
7FB9 2ØØ8            JR      NZ,NTENDF       ;NOT END DEFINITION CHAR
7FBB 3A8Ø38          LD      A,(SHIFT)       ;SHIFT KEY PRESSED?
7FBE B7              OR      A               ;NOT ZERO IF IT IS
7FBF 2ØØ9            JR      NZ,ENDDEF       ;YES, END DEFINITION
7FC1 3E1F            LD      A,DEFKEY        ;RESTORE A
7FC3 77      NTENDF  LD      (HL),A          ;ADD CHAR TO USER STRING
```

```
7FC4 23       02270          INC     HL              ;BUMP POINTER TO USER STRING
7FC5 CD3A03   02280          CALL    PUTC            ;ECHO KEY PRESSED
7FC8 10E3     02290          DJNZ    GETC            ;REPEAT IF MAX STRING
              02300                                  ; LENGTH NOT YET EXCEEDED
7FCA AF       02310 ENDDEF   XOR     A               ;PUT NULL END-OF-STRING
7FCB 77       02320          LD      (HL),A          ; MARKER AFTER USER STRING
7FCC 21EF7F   02330          LD      HL,ENDMSG       ;PUT OUT 'END OF DEF' MSG
7FCF CDD57F   02340          CALL    PUTSTR          ;ALSO LEAVES Ø IN A
7FD2 E1       02350          POP     HL              ;RESTORE CALLER'S HL
7FD3 C1       02360          POP     BC              ;RESTORE CALLER'S BC
7FD4 C9       02370          RET                     ;AND RETURN TO CALLER
              02380 ;
              02390 ;PUTSTR: PUT STRING AT (HL) ON SCREEN.  STRING IS
              02400 ;        TERMINATED BY A Ø BYTE.
              02410 ;
7FD5 7E       02420 PUTSTR   LD      A,(HL)          ;GET CHARACTER FROM STRING
7FD6 B7       02430          OR      A               ;IF CHARACTER IS NULL (Ø)
7FD7 C8       02440          RET     Z               ;THEN FINISHED
7FD8 CD3A03   02450          CALL    PUTC            ;ELSE PUT CHAR ON SCREEN
7FDB 23       02460          INC     HL              ;POINT AT NEXT CHARACTER
7FDC 18F7     02470          JR      PUTSTR          ;AND GO GET IT
              02480 ;
              02490 ;PROMPT MESSAGES:
7FDE ØD       02500 STRTDF   DEFB    ØDH             ;CARRIAGE RETURN
7FDF 44       02510          DEFM    'DEFINE STRING:'
7FED ØD       02520          DEFB    ØDH
7FEE ØØ       02530          DEFB    Ø               ;END OF STRTDF
7FEF ØD       02540 ENDMSG   DEFB    ØDH             ;CARRIAGE RETURN
7FFØ 45       02550          DEFM    'END DEFINITION'
7FFE ØD       02560          DEFB    ØDH
7FFF ØØ       02570          DEFB    Ø               ;END OF ENDMSG
7E58          02580          END     INIT
ØØØØØ TOTAL ERRORS


CALL    ØØCD ØØ19Ø   Ø131Ø Ø214Ø
DEF     7FAØ Ø206Ø   Ø201Ø
DEFINE  7F96 Ø199Ø   Ø143Ø
DEFKEY  ØØ1F ØØ2ØØ   Ø142Ø Ø202Ø Ø22ØØ Ø225Ø
ENDDEF  7FCA Ø231Ø   Ø224Ø
ENDMSG  7FEF Ø254Ø   Ø233Ø
GETC    7FAD Ø212Ø   Ø219Ø Ø229Ø
INIT    7E58 ØØ29Ø   Ø258Ø
KEYDR1  7F4E Ø132Ø   ØØ3ØØ
KEYDR2  7FBØ Ø215Ø   ØØ31Ø
KEYDRV  4Ø16 ØØ14Ø   ØØ29Ø ØØ34Ø
KEYWRD  7F6C Ø164Ø   Ø157Ø
LA      7EAE ØØ7ØØ   Ø164Ø
LB      7EB5 ØØ72Ø
LC      7EBA ØØ74Ø
LD      7ECØ ØØ76Ø
LE      7EC5 ØØ78Ø
LF      7ECD ØØ8ØØ
LG      7ED1 ØØ82Ø
LH      7ED6 ØØ84Ø
LI      7EDB ØØ86Ø
LJ      7EE2 ØØ88Ø
LK      7EE7 ØØ9ØØ
LL      7EEE ØØ92Ø
LM      7EF3 ØØ94Ø
LN      7EF8 ØØ96Ø
LO      7EFD ØØ98Ø
LP      7FØ2 Ø1ØØØ
LQ      7FØ8 Ø1Ø2Ø
LR      7FØF Ø1Ø4Ø
LS      7F16 Ø1Ø6Ø
LT      7F1C Ø1Ø8Ø
LU      7F21 Ø11ØØ
LV      7F27 Ø112Ø
LW      7F3Ø Ø114Ø
LX      7F36 Ø116Ø
LY      7F3B Ø118Ø
LZ      7F4Ø Ø12ØØ
NOTEND  7F9Ø Ø192Ø   Ø19ØØ
NTENDF  7FC3 Ø226Ø   Ø221Ø
NXTC    7F75 Ø17ØØ   Ø173Ø Ø175Ø
```

Program continued

```
ONESTR 7F47 Ø125Ø   ØØ33Ø
OSFLAG 7EAB ØØ57Ø   Ø125Ø Ø181Ø Ø191Ø
OSPTR  7EAC ØØ6ØØ   Ø179Ø Ø187Ø Ø193Ø
PUTC   Ø33A ØØ18Ø   Ø228Ø Ø245Ø
PUTSTR 7FD5 Ø242Ø   Ø2Ø9Ø Ø234Ø Ø247Ø
REST   7E6F ØØ54Ø
SETMD  7F7F Ø18ØØ
SHIFT  388Ø ØØ16Ø   Ø199Ø Ø222Ø
SRCHDN 7F7C Ø179Ø   Ø159Ø Ø168Ø
STRTDF 7FDE Ø25ØØ   Ø2Ø8Ø
SUBMOD 7F62 Ø155Ø   Ø145Ø Ø149Ø
SUBST  7F85 Ø186Ø   Ø127Ø
UDSKEY ØØ1A ØØ21Ø   Ø144Ø Ø156Ø
USED   ØØØ5 ØØ53Ø   ØØ54Ø
USTLEN ØØ4Ø ØØ23Ø   ØØ54Ø Ø211Ø
USTR   7E6A ØØ5ØØ   ØØ53Ø Ø158Ø Ø21ØØ
```

UTILITY

BREAK Disable

by Jim Rastin

Have you ever accidentally pressed the BREAK key during the execution of a BASIC program and fumed because you lost your place? Have you ever had someone play with your TRS-80 during a demonstration and had to post a sign DO NOT TOUCH BREAK KEY? I have, and to prevent this from happening, I wrote this program.

Although after pressing BREAK you can type CONT without losing your variables, you may lose the display unless you rerun the program. But this costs you your variables! My program eliminates the effect of the BREAK key when running a program.

Program Listing 1 is written in BASIC, although when executed it POKEs into memory a machine-language program (see Program Listing). But you need not know machine language to use it. Reserve 32742 when you first power up and see MEMORY SIZE.

The program has some built-in advantages. It acts as a debounce program. Typing POKE 32763,201 restores the BREAK key so that if you use auto numbering to write a program, you may BREAK out of this mode.

Type POKE 32763,192 and the BREAK key no longer functions. Typing POKE 32756,x (where x is a number from 1 to 255) will slow computer operation. The higher the number POKEd into 32756, the slower the TRS-80 functions, as it delays for every execution performed. For example, if you POKE 32756 with 255, it takes a long time to enter any letter from the keyboard. It also runs your programs very slowly. Typing POKE 32756, 0 returns your TRS-80 to normal speed. (The slow speed works well for debugging programs or watching a list appear on the screen.)

You can either run the program as shown and CLOAD your programs, or you can type the program omitting lines 50, 60, and 70 and place it in front of your own. It is then incorporated into one program containing all the features mentioned above.

You can vary the amount of time the debounce delay takes by typing POKE 32755, x (where x is a number from 1 to 200). The program first loads this memory location with 50. One word of caution—don't load this memory location with less than two. At a setting of less than ten the debounce won't be very effective.

Program Listing 1. *BASIC*

Encyclopedia Loader™

```
10 CLS :
   FOR X = 32743 TO 32767
20  READ A
30  POKE X,A:
    NEXT
40 POKE 16526,231:
   POKE 16527,127:
   X = USR(0)
50 PRINT @ 512,"DEBOUNCE OPERATIONAL WITHOUT BREAK KEY";
60 FOR X = 1 TO 1000:
    NEXT
70 NEW
80 DATA 33,238,127,34,22,64,201,205,227,3,103,1,50,0,205,96,0,124,2
   54,1,192,62,0,201,0,0
```

Program Listing 2. *Machine language*

```
00100 LD HL,7FEE           ;KEYBOARD DRIVER ADD
00110 LD (4016),HL
00120 RET
00130 CALL 03E3H           ;CALL KEYBOARD DRIVER
00140 LD H,A               ;SAVE A
00150 LD BC,0050           ;DEBOUNCE DELAY
00160 CALL 0060H
00170 LD A,H               ;RESTORE A
00180 CP 1                 ;COMPARE BREAK
00190 RET NZ               ;RET UNLESS BREAK
00200 LD A,0               ;PUT 0 IN BREAK
00210 RET                  RETURN
00220 NOP
*
*
*_
```

Z-80 Disassembler

by Daniel Lovy

Machine-language programs are faster, more efficient, and generally more customized than their higher-level language counterparts. Unfortunately they are almost totally undigestable by anyone except the machine.

An assembler can eat a readable assembly-language program and spit back a stream of seemingly unrelated numbers. This program (see the Program Listing) is a Z-80 disassembler which takes these numbers and the object code and regenerates the assembly-language mnemonics. Written in TRS-80 Level II BASIC, it runs in about 10K of memory and can disassemble the entire Z-80 instruction set.

The program begins by asking which base you want to work with (hex or decimal)—the default is hex. Next, it asks you to decide between an ASCII dump or using the disassembler—the default is the disassembler. Then it asks you if you want to have the output sent to a printer—no is the default. The program then asks for the starting address. When disassembling something in RAM, be sure to set the memory size so the program does not destroy it. Hitting ENTER three times will default to the disassembler in hex with no printer output. If you want the program to stop and take up its work at another address, type I for interrupt.

Program Organization

The lines up to 160 initialize everything and take care of the base conversions. The first number is then PEEKed from memory and converted to binary. Line 210 tests to see if it is the first byte of a multiple byte op-code. If it is not, the program looks at the first two bits. These determine which of four types of instructions the op-code could be: a miscellaneous one, a load, an arithmetic and logic, or a jump call and return. From there the program uses one or more of the following subroutines to complete the translation:

- 2150—decodes the register pair
- 2230—decodes the flag conditions
- 2340—fetches data
- 2040—decodes the register involved

The program keeps track of how many bytes the instruction takes and adds it to the program counter. The scheme is much the same for the multiple-byte op-codes. There are a few additional subroutines worth mentioning:

● 2400—does the conversion to binary
● 2410—translates the two's complement numbers used in relative addresses into the proper positive and negative decimal numbers
● 2500—this is a two-line decimal to hex converter

At line 4000, if you request output to the printer, PR = 1 and is POKED into location 16540. Then the next print statement executed will be sent to the line printer. The program resets this after each print, which is why I made this a subroutine.

If you do not want to key in this entire program at once or would like a shorter version, you could replace line 1100 with:

```
1100 O1$ = "MULTIPLE BYTE OP—CODE": INC = 4: GOTO 240
```

Lines 1110-2030 may then be omitted. This saves nearly 100 lines and about 3K of memory, but the program will be unable to decipher the multiple-byte op-codes. This does not occur often, and you can add these lines later.

```
0000  F3       DI
0001  AF       XOR     A
0002  C37406   JP      0674
0005  C30040   JP      4000
0008  C30040   JP      4000
000B  E1       POP     HL
000C  E9       JP      (HL)
000D  C39F06   JP      069F
0010  C30340   JP      4003
0013  C5       PUSH    BC
0014  0601     LD     B,0001
0016  182E     JR      46
0018  C30640   JP      4006
001B  C5       PUSH    BC
001C  0602     LD     B,0002
001E  1826     JR      38
0020  C30940   JR      4009
0023  C5       PUSH    BC
```

Figure 1. *Sample output*

Program Listing. *Z-80 Disassembler*

Encyclopedia Loader

```
10 :
   ' ****     Z-80 DISASSEMBLER     ****
20 :
   ' ****        DANIEL LOVY        ****
30 :
   ' ****      JANUARY 1981         ****
40 CLS :
   CLEAR 200:
   DEFINT Z,P:
   PRINT TAB(22)"Z-80 DISASSEMBLER":
   PRINT :
   PRINT
50 DIM A(8),HEX$(15)
60 FOR Z = 0 TO 9:
     HEX$(Z) = CHR$(Z + 48):
     NEXT :
   FOR Z = 65 TO 70:
     HEX$(Z - 55) = CHR$(Z):
     NEXT
70 LE = 4
80 PRINT "DO YOU WISH TO WORK IN DECIMAL OR HEXADECIMAL (D/H)":
   INPUT A$:
   IF A$ < > "D"
    THEN
      H = 1
90 IF H = 1
    THEN
      A$ = "HEX" :
    ELSE
      A$ = "DECIMAL"
95 INPUT "DO YOU WANT TO USE THE DISASSEMBLER OR GET AN ASCII DUMP
   (D/A)  ";CH$
97 PRINT "DO YOU WANT THE OUTPUT TO GO TO THE PRINTER (Y/N)":
   INPUT AN$:
   IF LEFT$(AN$,1) = "Y"
    THEN
      PR = 1
100 PC$ = "":
    PC = 0:
    PRINT "ENTER STARTING ADDRESS (IN ";A$;")":
    INPUT PC$
110 IF PC$ = ""
     THEN
       100
120 IF H < > 1
     THEN
       PC = VAL(PC$):
       GOTO 165
130 FOR Z = LEN(PC$) TO 1 STEP - 1:
      TEM$ = MID$(PC$,Z,1)
140   FOR Z1 = 0 TO 15:
        IF TEM$ = HEX$(Z1)
         THEN
           PC = Z1 * 16 [ ( LEN(PC$) - Z) + .2 + PC:
           GOTO 150:
         ELSE
           NEXT Z1:
         PRINT "INVALID ENTRY":
         GOTO 100
150   NEXT Z
160 PRINT :
    PRINT
165 IF CH$ = "A"
     THEN
       3000
170 WR = PC:
    BYT = PEEK(PC)
```

Program continued

```
180 01$ = "":
    02$ = "":
    0T$ = ""
190 TEM = BYT:
    GOSUB 2400
200 :
    ' ***                        TEST FOR MULTIBLE BYTES
210 IF BYT = 221 OR BYT = 203 OR BYT = 237 OR BYT = 253
     THEN
      FL = 1:
      IX$ = "HL":
      GOTO 1100 :
     ELSE
      FL = 0
220 :
    ' ***                        SEPARATE ONE BYTE GROUPS

230 ON A(8) * 2 + A(7) + 1 GOSUB 290,590,680,830
240 GOSUB 4000:
    IF H = 1
     THEN
      CNV = WR:
      GOSUB 2500:
      PRINT CNV$;"  "; :
     ELSE
      PRINT WR;" ";
250 PC = PC + INC:
    FOR Z3 = WR TO PC - 1:
     TEM = PEEK(Z3)
260  GOSUB 4000:
     LE = 2:
     IF H = 1
      THEN
       CNV = TEM:
       GOSUB 2500:
       PRINT CNV$; :
      ELSE
       PRINT TEM;
270  NEXT Z3:
    LE = 4
280 GOSUB 4000:
    PRINT TAB(23)01$;"    ";02$:
    IN$ = INKEY$:
    IF IN$ = "I"
     THEN
      100 :
     ELSE
      GOTO 170
290 :
    ' ***                        MISC OP CODES

300 ON A(3) * 4 + A(2) * 2 + A(1) + 1 GOSUB 320,420,450,520,540,550,
    560,570
310 RETURN
320 ON A(6) * 4 + A(5) * 2 + A(4) + 1 GOTO 330,340,350,360,370,380,3
    90,400:
    '    NOTE, THIS IS A GOTO
330 INC = 1:
    01$ = "NOP":
    RETURN
340 INC = 1:
    01$ = "EX":
    02$ = "AF,AF'":
    RETURN
350 INC = 2:
    01$ = "DJNZ":
    GOSUB 2330:
    COM = D1:
    GOSUB 2420:
    02$ = STR$(COM):
    RETURN
```

```
360 GOSUB 410:
    O2$ = D1$:
    RETURN
370 GOSUB 410:
    O2$ = "NZ," + D1$:
    RETURN
380 GOSUB 410:
    O2$ = "Z," + D1$:
    RETURN
390 GOSUB 410:
    O2$ = "NC," + D1$:
    RETURN
400 GOSUB 410:
    O2$ = "C," + D1$:
    RETURN
410 INC = 2:
    O1$ = "JR":
    GOSUB 2230:
    GOSUB 2330:
    COM = D1:
    GOSUB 2420:
    D1$ = STR$(COM):
    O2$ = OT$ + "," + D1$:
    RETURN
420 IF A(4) = 1
     THEN
      440
430 INC = 3:
    O1$ = "LD":
    GOSUB 2160:
    GOSUB 2330:
    O2$ = OT$ + "," + D3$:
    RETURN
440 INC = 1:
    O1$ = "ADD":
    GOSUB 2160:
    O2$ = "HL," + OT$:
    RETURN
450 O1$ = "LD":
    IF A(6) = 1
     THEN
      480
460 INC = 1:
    GOSUB 2160:
    IF A(4) = 0
     THEN
      O2$ = "(" + OT$ + "),A":
      RETURN
470 O2$ = "A,(" + OT$ + ")":
    RETURN
480 INC = 3:
    GOSUB 2330:
    IF A(4) = 0
     THEN
      O2$ = "(" + D3$ + ")," :
     ELSE
      500
490 IF A(5) = 0
     THEN
      O2$ = O2$ + "HL":
      RETURN :
     ELSE
      O2$ = O2$ + "A":
      RETURN
500 IF A(5) = 1
     THEN
      O2$ = "A," :
     ELSE
      O2$ = "HL,"
510 O2$ = O2$ + "(" + D3$ + ")":
    RETURN
```

Program continued

```
520 INC = 1:
    GOSUB 2160:
    IF A(4) = 0
     THEN
      01$ = "INC" :
     ELSE
      01$ = "DEC"
530 02$ = OT$:
    RETURN
540 INC = 1:
    S1 = 6:
    S2 = 5:
    S3 = 4:
    GOSUB 2050:
    01$ = "INC":
    02$ = OT$:
    RETURN
550 INC = 1:
    S1 = 6:
    S2 = 5:
    S3 = 4:
    GOSUB 2050:
    01$ = "DEC":
    02$ = OT$:
    RETURN
560 INC = 2:
    S1 = 6:
    S2 = 5:
    S3 = 4:
    01$ = "LD":
    GOSUB 2050:
    GOSUB 2330:
    02$ = OT$ + "," + D1$:
    RETURN
570 INC = 1:
    FOR Z = 1 TO A(6) * 4 + A(5) * 2 + A(4) + 1:
     READ 01$:
     NEXT :
    RESTORE :
    RETURN
580 DATA RLCA,RRCA,RLA,RRA,DAA,CPL,SCF,CCF
590 :
    ' ***                         ONE BYTE LOAD GROUP

600 01$ = "LD"
610 S1 = 6:
    S2 = 5:
    S3 = 4
620 GOSUB 2050
630 02$ = OT$
640 S1 = 3:
    S2 = 2:
    S3 = 1:
    GOSUB 2050
650 02$ = 02$ + "," + OT$
660 INC = 1
670 RETURN
680 :
    ' ***                                ARITHM. & LOGIC

690 INC = 1
700 ON A(6) * 4 + A(5) * 2 + A(4) + 1 GOTO 710,720,730,740,750,760,7
    70,780
710 01$ = "ADD":
    02$ = "A":
    GOTO 790
720 01$ = "ADC":
    02$ = "A":
    GOTO 790
730 01$ = "SUB":
    GOTO 790
```

```
740 O1$ = "SBC":
    O2$ = "A":
    GOTO 790
750 O1$ = "AND":
    GOTO 790
760 O1$ = "XOR":
    GOTO 790
770 O1$ = "OR":
    GOTO 790
780 O1$ = "CP"
790 IF INC = 2
     THEN
      RETURN :
     ELSE
      S1 = 3:
      S2 = 2:
      S3 = 1:
      GOSUB 2050
800 IF O2$ = ""
     THEN
      O2$ = O2$ + OT$:
      GOTO 820
810 O2$ = O2$ + "," + OT$
820 RETURN
830 :
    ' ***                                    JP CALL & RET

840 IF FL = 1
     THEN
      RG$ = IX$ :
     ELSE
      RG$ = "HL"
850 ON A(3) * 4 + A(2) * 2 + A(1) + 1 GOSUB 870,880,940,950,1040,105
    0,1070,1080
860 RETURN
870 INC = 1:
    O1$ = "RET":
    GOSUB 2220:
    O2$ = OT$:
    RETURN
880 INC = 1:
    IF A(4) = 0
     THEN
      O1$ = "POP":
      GOSUB 2160:
      O2$ = OT$:
      RETURN
890 ON A(6) * 2 + A(5) + 1 GOTO 900,910,920,930
900 O1$ = "RET":
    RETURN
910 O1$ = "EXX":
    RETURN
920 O1$ = "JP":
    O2$ = "(" + RG$ + ")":
    RETURN
930 O1$ = "LD":
    O2$ = "SP," + RG$:
    RETURN
940 INC = 3:
    O1$ = "JP":
    GOSUB 2220:
    GOSUB 2330:
    O2$ = OT$ + "," + D3$:
    RETURN
950 ON A(6) * 4 + A(5) * 2 + A(4) + 1 GOTO 960,970,980,990,1000,1010
    ,1020,1030
960 INC = 3:
    O1$ = "JP":
    GOSUB 2330:
    O2$ = D3$:
    RETURN
```

Program continued

```
 970 RETURN
 980 INC = 2:
     O1$ = "OUT":
     GOSUB 2330:
     O2$ = "(" + D1$ + "),A":
     RETURN
 990 INC = 2:
     O1$ = "IN":
     GOSUB 2330:
     O2$ = "A,(" + D1$ + ")":
     RETURN
1000 INC = 1:
     O1$ = "EX":
     O2$ = "(SP)," + RG$:
     RETURN
1010 INC = 1:
     O1$ = "EX":
     O2$ = "DE,HL":
     RETURN
1020 INC = 1:
     O1$ = "DI":
     RETURN
1030 INC = 1:
     O1$ = "EI":
     RETURN
1040 INC = 3:
     O1$ = "CALL":
     GOSUB 2220:
     GOSUB 2330:
     O2$ = OT$ + "," + D3$:
     RETURN
1050 IF BYT = 205
      THEN
       INC = 3:
       O1$ = "CALL":
       GOSUB 2330:
       O2$ = D3$:
       RETURN
1060 INC = 1:
     O1$ = "PUSH":
     GOSUB 2160:
     O2$ = OT$:
     RETURN
1070 INC = 2:
     GOSUB 700:
     GOSUB 2330:
     O2$ = O2$ + "," + D1$:
     RETURN
1080 INC = 1:
     O1$ = "RST":
     O2$ = STR$(A(6) * 32 + A(5) * 16 + A(4) * 8) + "D":
     RETURN
1090 :
     ' ***                            MULT. BYTE OP CODES

1100 ON A(8) * 8 + A(7) * 4 + A(6) * 2 + A(5) - 11 GOSUB 1130,1300,17
     20,1310
1110 O2$ = OT$:
     GOTO 240
1120 :
     ' ***                   TWO BYTE PO CODES      CB

1130 GOSUB 2330:
     PC = PC + 1:
     INC = 1:
     TEM = D1:
     GOSUB 2400
1140 S1 = 3:
     S2 = 2:
     S3 = 1:
     GOSUB 2050
```

```
1150 ON A(8) * 2 + A(7) + 1 GOSUB 1170,1260,1270,1280
1160 RETURN
1170 ON A(6) * 4 + A(5) * 2 + A(4) + 1 GOTO 1180,1190,1200,1210,1220,
     1230,1240,1250
1180 01$ = "RLC":
     RETURN
1190 01$ = "RRC":
     RETURN
1200 01$ = "RL":
     RETURN
1210 01$ = "RR":
     RETURN
1220 01$ = "SLA":
     RETURN
1230 01$ = "SRA":
     RETURN
1240 01$ = "NON EXSISTANT CODE":
     RETURN
1250 01$ = "SRL":
     RETURN
1260 01$ = "BIT":
     OT$ = STR$(A(6) * 4 + A(5) * 2 + A(4)) + "," + OT$:
     RETURN
1270 01$ = "RES":
     OT$ = STR$(A(6) * 4 + A(5) * 2 + A(4)) + "," + OT$:
     RETURN
1280 01$ = "SET":
     OT$ = STR$(A(6) * 4 + A(5) * 2 + A(4)) + "," + OT$:
     RETURN
1290 :
     ' ***                                OP CODES       DD & FD

1300 IX$ = "IX":
     GOTO 1320
1310 IX$ = "IY"
1320 GOSUB 2330:
     IF D1 = 203
      THEN
       1640
1330 GOSUB 2330:
     PC = PC + 1:
     TEM = D1:
     GOSUB 2400:
     ON A(8) * 2 + A(7) + 1 GOSUB 1350,1560,1600,1620
1340 RETURN
1350 ON A(3) * 4 + A(2) * 2 + A(1) + 1 GOSUB 1370,1380,1450,1480,1500
     ,1520,1540,1370
1360 RETURN
1370 INC = 1:
     01$ = "NON CODE":
     RETURN
1380 IF A(4) = 0
      THEN
       01$ = "LD":
       INC = 3:
       GOSUB 2330:
       OT$ = IX$ + "," + D3$:
       RETURN
1390 01$ = "ADD":
     INC = 1:
     OT$ = IX$ + ","
1400 A = A(6) * 2 + A(5):
     IF A = 0
      THEN
       OG$ = "BC"
1410 IF A = 1
      THEN
       OG$ = "DE"
1420 IF A = 2
      THEN
       OG$ = IX$
```

Program continued

```
1430 IF A = 3
      THEN
       OG$ = "SP"
1440 OT$ = OT$ + OG$:
     RETURN
1450 INC = 3:
     GOSUB 2330:
     O1$ = "LD"
1460 IF A(4) = 0
      THEN
       OT$ = "(" + D3$ + ")," + IX$ :
      ELSE
       OT$ = IX$ + ",(" + D3$ + ")"
1470 RETURN
1480 INC = 1:
     OT$ = IX$:
     IF A(4) = 0
      THEN
       O1$ = "INC" :
      ELSE
       O1$ = "DEC"
1490 RETURN
1500 INC = 2:
     O1$ = "INC":
     GOSUB 2330:
     COM = D1:
     GOSUB 2420
1510 OT$ = "(" + IX$ + "+" + STR$(COM) + ")":
     RETURN
1520 INC = 2:
     O1$ = "DEC":
     GOSUB 2330:
     COM = D1:
     GOSUB 2420
1530 OT$ = "(" + IX$ + "+" + STR$(COM) + ")":
     RETURN
1540 INC = 3:
     O1$ = "LD":
     GOSUB 2330:
     COM = D1:
     GOSUB 2420
1550 OT$ = "(" + IX$ + "+" + STR$(COM) + ")," + D2$:
     RETURN
1560 INC = 2:
     IF A(3) * 4 + A(2) * 2 + A(1) < > 6 AND A(6) * 4 + A(5)
     * 2 + A(4) < > 6
      THEN
       O1$ = "NON CODE":
       OT$ = "":
       INC = 1:
       RETURN
1570 O1$ = "LD":
     GOSUB 2330:
     IF A(3) * 4 + A(2) * 2 + A(1) = 6
      THEN
       1590
1580 S1 = 3:
     S2 = 2:
     S3 = 1:
     GOSUB 2050:
     COM = D1:
     GOSUB 2420:
     OT$ = "(" + IX$ + "+" + STR$(COM) + ")," + OT$:
     RETURN
1590 S1 = 6:
     S2 = 5:
     S3 = 4:
     GOSUB 2050:
     COM = D1:
     GOSUB 2420:
     OT$ = OT$ + ",(" + IX$ + "+" + STR$(COM) + ")":
      RETURN
1600 INC = 2:
     IF A(3) * 4 + A(2) * 2 + A(1) < > 6
      THEN
       O1$ = "NON CODE":
       INC = 1:
       OT$ = "":
       RETURN
1610 GOSUB 700:
     COM = D1:
     GOSUB 2420:
     OT$ = O2$ + " (" + IX$ + "+" + STR$(COM) + ")":
     RETURN
1620 INC = 1:
     IF A(6) * 4 + A(5) * 2 + A(4) < 4
      THEN
       O1$ = "NON CODE":
       RETURN
1630 GOSUB 830:
     OT$ = O2$:
     INC = 1:
     RETURN
1640 :
     ' ***                        THREE BYT OP CODES


1650 PC = PC + 1:
     GOSUB 2330:
     INC = 3:
     TEM = D2:
     GOSUB 2400
1660 ON A(8) * 2 + A(7) + 1 GOSUB 1170,1260,1270,1280
1670 COM = D1:
     GOSUB 2420:
     D$ = STR$(COM)
1680 OT$ = "(" + IX$ + "+" + D$ + ")"
1690 TEM = D2:
     GOSUB 2400
1700 IF A(8) + A(7) = 0
      THEN
       RETURN
1710 OT$ = STR$(A(6) * 4 + A(5) * 2 + A(4)) + "," + OT$:
     RETURN
1720 GOSUB 2330:
     PC = PC + 1:
     INC = 1:
     TEM = D1:
     GOSUB 2400
1730 IF A(8) = 1
      THEN
       1920
1740 ON A(3) * 4 + A(2) * 2 + A(1) + 1 GOSUB 1760,1770,1780,1800,1830
     ,1840,1850,1860
1750 O2$ = OT$:
     RETURN
1760 INC = 1:
     S1 = 6:
     S2 = 5:
     S3 = 4:
     GOSUB 2050:
     O1$ = "IN":
     OT$ = OT$ + ",(C)":
     RETURN
1770 INC = 1:
     S1 = 6:
     S2 = 5:
     S3 = 4:
     GOSUB 2050:
     O1$ = "OUT":
     OT$ = "(C)," + OT$:
     RETURN
```

Program continued

```
1780 INC = 1:
     IF A(4) = 0
      THEN
       01$ = "SBc" :
      ELSE
       01$ = "ADC"
1790 GOSUB 2160:
     OT$ = "HL," + OT$:
     RETURN
1800 GOSUB 2330:
     GOSUB 2160:
     INC = 3:
     01$ = "LD"
1810 IF A(4) = 0
      THEN
       OT$ = "(" + D3$ + ")," + OT$:
       RETURN
1820 OT$ = OT$ + ",(" + D3$ + ")":
     RETURN
1830 INC = 1:
     OT$ = "":
     01$ = "NEG":
     RETURN
1840 OT$ = "":
     INC = 1:
     IF A(4) = 0
      THEN
       01$ = "RETN":
       RETURN :
      ELSE
       01$ = "RETI":
       RETURN
1850 INC = 1:
     OT$ = "":
     01$ = "IM" + STR$(A(5) + A(4)):
     RETURN
1860 INC = 1:
     ON A(6) * 4 + A(5) * 2 + A(4) + 1 GOTO 1880,1870,1890,1870,1900,
     1910
1870 01$ = "NON CODE":
     OT$ = "":
     RETURN
1880 01$ = "LD":
     OT$ = "I,A":
     RETURN
1890 01$ = "LD":
     OT$ = "A,I":
     RETURN
1900 01$ = "RRD":
     OT$ = "":
     RETURN
1910 01$ = "RLD":
     OT$ = "":
     RETURN
1920 INC = 1:
     IF A(6) * 4 + A(5) * 2 + A(4) < 4
      THEN
       01$ = "NON CODE":
       OT$ = "":
       RETURN
1930 IF A(3) * 4 + A(2) * 2 + A(1) > 3
      THEN
       01$ = "NON CODE":
       OT$ = "":
       RETURN
1940 ON A(3) * 4 + A(2) * 2 + A(1) + 1 GOTO 1950,1960,1970,1980
1950 01$ = "LD":
     GOSUB 1990:
     01$ = 01$ + OG$:
     RETURN
1960 01$ = "CP":
     GOSUB 1990:
     01$ = 01$ + OG$:
     RETURN
1970 01$ = "IN":
     GOSUB 1990:
     01$ = 01$ + OG$:
     RETURN
1980 01$ = "OUT":
     GOSUB 1990:
     01$ = 01$ + OG$:
     RETURN
1990 OT$ = "":
     ON A(5) * 2 + A(4) + 1 GOTO 2000,2010,2020,2030
2000 OG$ = "I":
     RETURN
2010 OG$ = "D":
     RETURN
2020 OG$ = "IR":
     RETURN
2030 OG$ = "DR":
     RETURN
2040 :
     ' ***                              REGISTER DECODERS

2050 ON A(S1) * 4 + A(S2) * 2 + A(S3) + 1 GOSUB 2070,2080,2090,2100,2
     110,2120,2130,2140
2060 RETURN
2070 OT$ = "B":
     RETURN
2080 OT$ = "C":
     RETURN
2090 OT$ = "D":
     RETURN
2100 OT$ = "E":
     RETURN
2110 OT$ = "H":
     RETURN
2120 OT$ = "L":
     RETURN
2130 OT$ = "(HL)":
     RETURN
2140 OT$ = "A":
     RETURN
2150 :
     ' ***                      REGESTER PAIRS

2160 ON A(6) * 2 + A(5) + 1 GOSUB 2180,2190,2200,2210
2170 RETURN
2180 OT$ = "BC":
     RETURN
2190 OT$ = "DE":
     RETURN
2200 IF FL = 1
      THEN
       OT$ = IX$:
       RETURN :
      ELSE
       OT$ = "HL":
       RETURN
2210 OT$ = "SP":
     RETURN
2220 :
     ' ***                      FLAG CONDITIONS

2230 ON A(6) * 4 + A(5) * 2 + A(4) + 1 GOSUB 2250,2260,2270,2280,2290
     ,2300,2310,2320
2240 RETURN
2250 OT$ = "NZ":
     RETURN
2260 OT$ = "Z":
```

Program continued

```
      RETURN
2270 OT$ = "NC":
      RETURN
2280 OT$ = "C":
      RETURN
2290 OT$ = "PO":
      RETURN
2300 OT$ = "PE":
      RETURN
2310 OT$ = "P":
      RETURN
2320 OT$ = "M":
      RETURN
2330 :
     ' ***                              GET DATA BYTES

2340 D1 = PEEK(PC + 1):
     D2 = PEEK(PC + 2):
     D3 = (D2 * 256 + D1)
2350 IF H = 1
      THEN
       2370
2360 D1$ = STR$(D1):
     D2$ = STR$(D2):
     D3$ = STR$(D3):
     RETURN
2370 CNV = D1:
     GOSUB 2500:
     D1$ = CNV$:
     CNV = D2:
     GOSUB 2500:
     D2$ = CNV$
2380 CNV = D3:
     GOSUB 2500:
     D3$ = CNV$:
     RETURN
2390 :
     ' ***                         CONVERT TO BINARY

2400 FOR Z = 8 TO 1 STEP - 1:
      A(Z) = INT(TEM / 2 [ (Z - 1)):
      TEM = TEM - A(Z) * 2 [ (Z - 1):
      NEXT Z:
     RETURN
2410 :
     ' ***                         CONV TO TWOS COMP

2420 TEM = COM:
     GOSUB 2400
2430 IF A(8) = 0
      THEN
       RETURN
2440 COM = COM - 1:
     TEM = COM:
     GOSUB 2400
2450 FOR Z = 8 TO 1 STEP - 1:
      IF A(Z) = 1
       THEN
        A(Z) = 0 :
       ELSE
        A(Z) = 1
2460  NEXT Z:
     COM = 0
2470 FOR Z = 8 TO 1 STEP - 1:
      COM = COM + A(Z) * 2 [ (Z - 1):
      NEXT Z
2480 COM = - COM:
     RETURN
2490 :
     ' ***                         CONV TO HEX
```

```
2500 CNV$ = "":
     FOR Z = LE TO 1 STEP - 1:
      F = 16 [ (Z - 1):
      CNV$ = CNV$ + HEX$( INT(CNV / F))
2510  CNV = CNV - INT(CNV / F) * F:
      NEXT :
     RETURN
2590 :
     ' ***                         ASCII DUMPER

3000 CLS :
     PRINT PC$:
     BYT = PEEK(PC)
3010 GOSUB 4000:
     IF BYT < 32 OR BYT > 128
      THEN
       PRINT BYT; :
      ELSE
       PRINT CHR$(BYT);
3020 PC = PC + 1:
     BYT = PEEK(PC):
     GOTO 3010
4000 :
     ' ***                ROUTE OUTPUT TO EITHER SCREEN OR PRINTER

4010 POKE 16540,PR:
     RETURN
```

APPENDIX

APPENDIX

BASIC Program Listings

Debugging someone else's mistakes is no fun. In a business environment, where programs are continuously updated and programmers come and go, well-commented and structured programs are a must. Indeed, it behooves any serious programmer to learn structured technique.

The BASIC language has no inherent structure. Most interpreters allow remark lines and some are capable of ignoring unnecessary spacing, but BASIC is still more "Beginner's Instruction Code" than "All-purpose."

The listings in this encyclopedia are an attempt at formatting the TRS-80 BASICs. We think it makes them easier to read, easier to trace, and less imposing when it comes time to type them into the computer. You should *not*, however, type them in exactly as they appear. Follow normal syntax and entry procedures as described in your user's manual.

Level I Programs

Programs originally in Level I have been converted to allow running in Level II. To run in Level I, follow this procedure:

- Delete any dimension statements. Example: DIM A (25).
- Change PRINT@ to PRINTAT.
- Make sure that no INPUT variable is a STRING variable.
  Example: INPUT A$ would be changed to INPUT A and subsequent code made to agree.
- Abbreviate all BASIC statements as allowed by Level I.
  Example: *PRINT* is abbreviated *P*.

Model III Users

For the Model I, OUT255,0 and OUT255,4 turn the cassette motor off and on, respectively. For the Model III, change these statements to OUT236,0 and OUT236,2.

APPENDIX

Glossary

A

ac input module—I/O rack module which converts various ac signals originating in user switches to the appropriate logic level for use within the processor.

ac output module—I/O rack module which converts the logic levels of the processor to a usable output signal to control a user's ac load.

access time—the elapsed time between a request for data and the appearance of valid data on the output pins of a memory chip. Usually 200–450 nanoseconds for TRS-80 RAM.

accumulator—the main register(s) in a microprocessor used for arithmetic, shifting, logical, and other operations.

accuracy—generally, the quality or freedom from mistake or error; the extent to which the results of a calculation or a measurement approach the true value of the actual quantities.

acoustic coupler—a connection to a modem allowing signals to be transmitted through a regular telephone handset.

active elements—any generators of voltage or current in an impedance network; also known as active components.

adaptor—a device for connecting parts that will not mate; a device designed to provide a compatible connection between systems or subsystems.

A/D converter—analog to digital converter. See D/A converter.

add with carry—a machine-language instruction in which one operand is added to another, along with a possible carry from the previous (lower-order) add.

address—a code that specifies a register, memory location, or other data source or destination.

ALGOL—an acronym for ALGOrithmic Language. A very high-level language used in scientific applications, generally on large-scale computers.

algorithm—a predetermined process for the solution of a problem or completion of a task in a finite number of steps.

alignment—the process of adjusting components of a system for proper interrelationships, including adjustments and synchronization for the components in a system. For the TRS-80, this usually applies to cassette heads and disk drives.

alphanumerics—refer to the letters of the alphabet and digits of the number system, specifically omitting the characters of punctuation and syntax.

alternating current—ac. Electric current that reverses direction periodically, usually many times per second.

ALU—Arithmetic Logic Unit.

Ampere—the unit of electric current in the meter-kilogram-second system of units; defined in terms of the force of attraction between two parallel current conductors; 1 coulomb/second.

Ampere-turn—a unit of magnetomotive force defined as the force of a closed loop of one turn with a current of one ampere flowing through the loop.

analog—the representation of a physical variable by another variable insofar as the proportional relationships are the same over some specified range.

analog input module—an I/O rack module which converts an analog signal from a user device to a digital signal which may be processed by the processor.

analog output module—an I/O rack module which converts a digital signal from the processor into an analog output signal for use by a user device.

AND—a Boolean logic function. Two operators are tested and, if both are true, the answer is true. Truth is indicated by a high bit, or 1 in machine language, or a positive value in BASIC. If the operators are bytes or words, each element is tested separately. A bit-by-bit logical operation which produces a one in the result bit only if both operand bits are ones.

anode—in a semiconductor diode, the terminal toward which electrons flow from an external circuit; the positive terminal.

APL—a programming language; a popular and powerful high-level mathematical language with extensive symbol manipulation.

argument—any of the independent variables accompanying a command.

Arithmetic Logic Unit—ALU. The section of a microprocessor which performs arithmetic functions such as addition or subtraction and logic functions such as ANDing.

arithmetic shift—a type of shift in which an operand is shifted right or left with the sign bit being extended (right shift) or maintained (left shift).

array—a collection of data items arranged in a meaningful pattern such as rows and columns which allow the collection and retrieval of data.

ASCII—American Standard Code for Information Interchange. An almost universally accepted code (at least for punctuation and capital letters) where characters and printer commands are represented by numbers between 0 and 255 (base 10). The number is referred to as an ASCII code.

assembler—software that translates operational codes into their binary equivalents on a statement-for-statement basis.

assembly language—a symbolic computer language that is translated by an assembler program into machine language, the numeric codes that are equivalent to microprocessor instructions.

asynchronous—not related through repeating time patterns.

asynchronous shift register—a shift register which does not require a clock. Register segments are loaded and shifted only at data entry.

B

backup—1) refers to making copies of all software and data stored externally 2) having duplicate hardware available.

base—the starting point for representation of a number in written form, where numbers are expressed as multiples of powers of the base value.

BASIC—an acronym for Beginner's All-purpose Symbolic Instruction Code. Developed at Dartmouth College and similar to FORTRAN. The standard, high-level, interactive language for microcomputers.

batch processing—a method of computing in which many of the same types of jobs or programs are done in one machine run. For example, a programming class may type programs on data cards and turn them over to the computer operator. All the cards are put into the card reader, and the results of each person's program are returned later. This is contrasted with interactive computing.

baud—1) a unit of data transmission speed equal to the number of code elements (bits) per second 2) a unit of signaling speed equal to the number of discrete conditions or signal events per second.

baud rate—a measure of the speed at which serial data is transmitted electronically. The equivalent of bits per second (bps) in microcomputing.

benchmark—to test performance against a known standard.

BCD—binary coded decimal. The 4-bit binary notation in which individual decimal digits (0 through 9) are represented by 4-bit binary numerals; e.g., the number 23 is represented by 0010 0011 in the BCD notation.

bias—a dc voltage applied to a transistor control electrode to establish the desired operating point.

bidirectional bus—a bus structure used for the two-way transmission of signals, that is, both input and output.

bidirectional printer—a printer capable of printing both left-to-right and right-to-left. Data is prestored in a fixed-size buffer.

binary—a number system which uses only 0 and 1 as digits. It is the equivalent of base 2. Used in microcomputing because it is easy to represent 1s and 0s by high and low electrical signals.

binary digit—the two digits, zero and one, used in binary notation. Often shortened to bit.

binary point—the point, analagous to a decimal point, that separates the integer and fractional portions of a binary mixed number.

bipolar device—a device whose operation depends on the transport of holes and electrons, usually made of layers of silicon with differing electrical characteristics.

bi-stable—two-state

bit—an abbreviation for binary digit. A 0 or 1 in the binary number system. A single high or low signal in a computer.

bit position—the position of a binary digit within a byte or larger group of binary digits. Bit positions in the Model I, II, III, and Color Computer are numbered from right to left, zero through N. This number corresponds to the power of two represented.

Boolean algebra—a mathematical system of logic first identified by George Boole, a 19th century English mathematician. Routines are described by combinations of ANDs, ORs, XORs, NOTs, and IF-THENs. All computer functions are based upon these operators.

boot—short for bootstrap loader or the use of one. The bootstrap loader is a very short routine whose purpose is to load a more sophisticated system into the computer when it is first turned on. On some machines it is keyed in, and on others it is in read only memory (ROM). Using this program is called booting or cold-starting the system.

borrow—one bit subtracted from the next higher bit position.

bps—bits per second.

breakdown—a large, abrupt rise in electric current due to decreased resistance in a semiconductor device caused by a small increase in voltage.

buffer—memory set aside temporarily for use by the program. Particularly refers to memory used to make up differences in the data transfer rates of the computer and external devices such as printers and disks.

bug—an error in software or hardware.

bump contact—a large area contact used for alloying directly to the substrate of a chip for mounting or interconnecting purposes.

bus—an ordered collection of all address, data, timing, and status lines in the computer.

byte—eight bits that are read simultaneously as a single code.

C

CAI—an acronym for Computer Aided Instruction.

card—a specially designed sheet of cardboard with holes punched in specific columns. The placement of the holes represents machine-readable data. Also a term referring to a printed circuit board.

card reader—a device for reading information from punched cards.

carrier—a steady signal that can be slightly modified (modulated) continuously. These modulations can be interpreted as data. In microcomputers the technique is used primarily in modem communications and tape input/output (I/O).

carry—a one bit added to the next higher bit position or to the carry flag.

carry flag—a bit in the microprocessor used to record the carry "off the end" as a result of a machine-language instruction.

cassette recorder—a magnetic tape recording and playback device for entering or storing programs.

cathode—in a semiconductor diode, the terminal from which electrons flow to an external circuit; the negative terminal.

character—a single symbol that is represented inside the computer by a specific code.

charge—a basic property of elementary particles of matter. The charge, measured in coulombs, is the algebraic sum of the electric charge of its constituents.

checksum—a method of detecting errors in a block of data by adding each piece of data in the block to a sum and comparing the final result to a predetermined result for the block of data.

chip—the shaped and processed semiconductor die mounted on a substrate to form a transistor or other semiconductor device.

circuit—a conductor or system of conductors through which an electric current may flow.

circuit card—a printed circuit board containing electronic components.

clear—to return a memory to a non-programmed state, usually represented as 0 or OFF (empty).

clobber—to destroy the contents of memory or a register.

clock—a simple circuit that generates the synchronization signals for the microprocessor. The speed or frequency of this clock directly affects the speed at which the computer can perform, regardless of the speed of which the individual chips are capable.

COBOL—COmmon Business-Oriented Language. A language used primarily for data processing. Allows programming statements that are very similar to English sentences.

Colossus—a British computer used to crack German Enigma codes during World War II.

common carrier—a communications transmission medium, such as the Direct Distance Dialing (DDD) network of the Bell System.

compiler—software that will convert a program written in a high-level language to binary code, on a many-for-one basis.

complement—a mathematical calculation. In computers it specifically refers to inverting a binary number. Any 1 is replaced by a 0, and vice versa.

complementary functions—two driving point functions whose sum is a positive constant.

complementary metal oxide semiconductor—CMOS. A signal inverting device formed by the combination of a p channel with an n channel device usually connected in series across the power supply.

complementary transistors—two transistors of opposite conductivity (pnp and npn) in the same functional unit.

computer interface—a device designed for data communication between a central computer and another unit such as a programmable controller processor.

concatenate—to put two things, each complete by itself, together to make a larger complete thing. In computers this refers to strings of characters or programs.

conditional jump—a machine-language instruction that jumps if a specified flag (or flags) is set or reset.

conductor—a substance, body, or other medium that is suitable to carry an electric current.

constant—a value that doesn't change.

control block—a storage area of a microprocessor containing the information required for control of a task, function, operation, or quantity of information.

coulomb—the unit of electric charge in SI units (International System of Units); the quantity of electric charge that passes any cross section of a conductor in one second when current is maintained constant at one ampere.

counter—in relay-panel hardware, an electro-mechanical device which can be wired and preset to control other devices according to the total cycles of one ON and OFF function. A counter is internal to the processor; i.e., it is controlled by a user-programmed instruction. A counter instruction has greater capability than any hardware counter.

CPU—central processing unit. The circuitry that actually performs the functions of the instruction set.

CRT—cathode ray tube. In computing this is just the screen the data appears on. A TV has a CRT.

cue—refers to positioning the tape on a cassette unit so that it is set up to a read/write section of tape.

current—the net transfer or electric charge per unit of time by free electrons; 1 ampere = 1 coulomb/second.

current mode logic—CML. Integrated circuit logic in which transistors are paralleled so as to eliminate current hogging.

cursor—a visual movable pointer used on a CRT by the programmer to indicate where an instruction is to be added to the program. The cursor is also used during editing functions.

cycle—a specific period of time, marked in the computer by the clock.

D

D/A converter—digital to analog converter. Common in interfacing computers to the outside world.

daisy chain—a bus line which interconnects devices for serial operation.

daisy wheel—a printer type which has a splined character wheel.

data—general term for numbers, letters, symbols, and analog quantities that serve as information for computer processing.

data base—refers to a series of programs each having a different function, but all using the same data. The data is stored in one location or file and each program uses it in a fashion that still allows the other program to use it.

data entry—the practice of entering data into the computer or onto a storage device. Knowledge of operating or programming a computer is not necessary for a data entry operator.

data link—equipment, especially transmission cables and interface modules, which permits the transmission of information.

debug—to remove bugs from a program.

decrement—to decrease the value of a number. In computers the number is in memory or a register, and the amount it is decremented is usually one.

dedicated—in computer terminology, a system set up to perform a single task.

default—that which is assumed if no specific information is given.

degauss—to demagnetize. Must be done periodically to tape and disk heads for reliable data transfer.

diagnostic program—a test program to help isolate hardware malfunctions in the programmable controller and application equipment.

die bond—a process in which chips are joined to a substrate.

differential discriminator—a circuit that passes only pulses whose amplitudes are between two predetermined values, neither of which are zero.

digital—the representation of data in binary code. In microcomputers, a high electrical signal is a 1 and a low signal is a 0.

digital circuit—an electronic network designed to respond at input voltages at one level, and similarly, to produce output voltages at one level.

diode—a device with an anode and a cathode which permits current flow in one direction and inhibits current flow in the other direction.

diode transistor logic—a circuit that uses diodes, transistors, and resistors to provide logic functions.

direct current—dc. Electric current which flows in only one direction; the term designates a practically non-pulsating current.

disassembly—remaking an assembly source program from a machine-code program.

disk—an oxide-coated, circular, flat object, in a variety of sizes and containers, on which computer data can be stored.

disk controller—an interface between the computer and the disk drive.

disk drive—a piece of hardware that rotates the disk and performs data transfer to and from the disk.

disk operating system—DOS. The system software that manipulates the data to be sent to the disk controller.

displacement—a signed value in machine language used in defining a memory address.

dividend—the number that is divided by the divisor. In A/B, A is the dividend.

divisor—the number that "goes into" the dividend in a divide operation. In A/B, B is the divisor.

DMA—direct memory access. A process where the CPU is disabled or bypassed temporarily and memory is read or written to directly.

documentation—a collection of written instructions necessary to use a piece of hardware, software, or a system.

domain—a region in a solid within which elementary atomic, molecular, magnetic, or electric moments are uniformly arrayed.

doping—the addition of impurities to a semiconductor to achieve a desired characteristic.

dot-matrix printer—instead of each letter having a separate type head (like a typewriter), a single print head makes the characters by printing groups of dots. The print is not as easy to read, but such printers are less expensive to manufacture.

double-dabble—a method of converting from binary to decimal representation by doubling the leftmost bit, adding the next bit, and continuing until the rightmost bit has been processed.

downtime—the time when a system is not available for production due to required maintenance.

driver—a small piece of system software used to control an external device such as a keyboard or printer.

dump—to write data from memory to an external storage device.

duplex—refers to two-way communications taking place independently, but simultaneously.

dynamic memory—circuits that require a periodic (every few milliseconds) recharge so that the stored data is not lost.

E

EAROM—an acronym for Electrical Alterable Read Only Memory. The chip can be read at normal speed, but must be written to with a slower process. Once written to, it is used like a ROM, but can be completely erased if necessary.

editor—a program that allows text to be entered into memory. Interactive languages usually have their own editors.

electron—a stable elementary particle with a negative electric charge of about -1.602×10^{-19} coulomb.

emitter-coupled logic—a form of current mode logic in which the emitters of two transistors are connected to a single current-carrying resistor in a way that only one transistor conducts at a time.

enhancement mode—operation of a field effect transistor in which no current flows when zero gate voltage is applied, and increasing the gate voltage increases the current.

EOF—End Of File.

EOL—End Of Line (of text).

EPROM—Erasable Programmable Read Only Memory. A read only memory in which stored data can be erased by ultraviolet light or other means and reprogrammed bit-by-bit with appropriate voltage pulses.

Exclusive OR—a bit-by-bit logical operation which produces a one bit in the result only if one or the other (but not both) operand bits is a one.

execution—the performance of a specific operation such as would be accomplished through processing one instruction, a series of instructions, or a complete program.

execution cycle—a cycle during which a single instruction of one specific operation.

execution time—the total time required for the execution to actually occur.

expansion interface—a device attached to the computer that allows a greater amount of memory or attachment of other peripherals.

exponent—the power of two of a floating-point number.

F

feedback—the signal or data fed back to the programmable controller from a controlled machine or process to denote its response to the command signal.

fetch cycle—a cycle during which the next instruction to be performed is read from memory.

Fibonacci series—the sequence of number 1, 1, 2, 3, 5, 8, 13, 21, 34,. . . in which each term is computed by addition of the two previous terms.

field-effect transistor—FET. A transistor in which the resistance of the current path from the source to drain is modulated by applying a transverse electric field between grid or gate electrodes; the electric field varies the thickness of depletion layers between the gates, thereby reducing the conductance.

file—a set of data, specifically arranged, that is treated as a single entity by the software or storage device.

filter—electrical device used to suppress undesirable electrical noise.

firmware—software that is made semi-permanent by putting it into some type of ROM.

flag—a single bit that is high (set) or low (reset), used to indicate whether or not certain conditions exist or have occurred.

flip chip—a tiny semiconductor die having terminations all on one side in the form of solder pads or bump contacts; after the surface of the chip has been passivated or otherwise treated, it is flipped over for attaching to a matching substrate.

flip-flop—a bi-stable device that assumes either of two possible states such that the transition between the states must be accomplished by electronic switching.

floating-point number—a standard way of representing any size of number in computers. Floating-point numbers contain a fractional portion (mantissa) and power of two (exponent) in a form similar to scientific notation.

flowcharting—a method of graphically displaying program steps, used to develop and define an algorithm before writing the actual code.

FORTRAN—FORmula TRANslator. One of the first high-level languages, written specifically to allow easy entry of mathematical problems.

full duplex—a mode of data transmission that is the equivalent of two paths—one in each direction simultaneously.

G

game theory—see von Neumann.

garbage—computer term for useless data.

gate—a circuit that performs a single Boolean function. A circuit having an output and a multiplicity of inputs, so designed that the output is energized only when a certain combination of pulses is present at the inputs.

GIGO—Garbage In, Garbage Out. One of the rules of computing. If the data going into the computer is bad, the data coming out will be bad also.

graphics—information displayed pictorially as opposed to alphanumerically.

ground—a conducting path, intentional or accidental, between an electric circuit or equipment and the earth, or some conducting body serving in place of the earth.

H

H—a suffix for hexadecimal, e.g., 4FFFH.

half duplex—data can flow in both directions, but not simultaneously. See duplex.

Hall effect—the development of a transverse electric field in a current-carrying conductor placed in a magnetic field; ordinarily the conductor is positioned so that the magnetic field is perpendicular to the direction of current flow and the electric field is perpendicular to both.

Hall generator—a generator using the Hall effect to give an output voltage proportional to magnetic field strength.

handshaking—a term used in data transfer. Indicates that beside the data lines there are also signal lines so both devices know precisely when to send or receive data. Handshaking requires clocking pulses on both ends of the communications line. Contrast with buffer.

hangup—the computer has ceased processing inexplicably.

hard copy—a printout; any form of printed document such as a ladder diagram, program listing, paper tape, or punched cards.

hard magnetic—a term describing a metal having a high coercive force which gives a high magnetic hysteresis; usually a permanent magnetic material.

hard wired—having a fixed wired program or control system built in by the manufacturer and not subject to change by programming.

hardware—refers to any physical piece of equipment in a computer system.

hex—hexadecimal.

hexa-dabble—conversion from hexadecimal to decimal by multiplying each hex digit by sixteen and adding the next hex digit until the last (rightmost) hex digit has been reached.

hexadecimal—representation of numbers in base sixteen by use of the hexadecimal digits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F.

high—a signal line logic level. The computer senses this level and treats it as a binary 1.

high-level language—a programming language which is CPU-independent and closely resembles English.

high order—see most significant bit.

HIT—acronym for Hash Index Table. A section of the directory on a TRS-80 disk.

hole—a mobile vacancy having an energy state near the top of the energy band of a solid; behaves as though it were a positively charged particle.

host computer—the primary computer in a multi-computer or terminal hookup.

human engineering—usually refers to designing hardware and software with ease of use in mind.

hysteresis—an oscillator effect wherein a given value of an operating parameter may result in multiple values of output power or frequency.

I

IC—integrated circuit.

immediate—addressing mode in which the address of the information that an operation is supposed to act upon immediately follows the operation code.

inclusive OR—a bit-by-bit logical operation which produces a one-bit result if one or the other operand bits, or both is a one.

increment—to increase, usually by one. See decrement.

indexed—addressing mode where the information is addressed by a specified value, or by the value in a specified register.

indirect—addressing mode in which the address given points to another address, and the second address is where the information actually is.

input devices—devices such as limit switches, pressure switches, push buttons, etc., that supply data to a programmable controller. These discrete inputs are two types: those with common return, and those with individual returns (referred to as isolated inputs). Other inputs include analog devices and digital encoders.

instruction—a command or order that will cause a computer to perform one certain prescribed operation.

insulator—a nonconducting material used for supporting or separating conductors to prevent undesired current flow to other objects.

integer variable—a BASIC variable type. It can hold values of −32,768 through +32,767 in two-byte two's complement notation.

integrated circuit—IC. An interconnected array of active and passive elements integrated with a single semiconductor substrate or deposited on the substrate and capable of performing at least one electronic circuit function. See chip.

integrated injection logic—I^2L. Integrated circuit logic which uses a simple and compact bipolar transistor gate structure which makes possible large scale integration on silicon for logic arrays and other analog and digital applications.

intelligent terminal—a terminal with a CPU and a certain amount of memory that can organize the data it receives and thus achieve a high level of handshaking with the host computer.

interactive computing—refers to the appearance of a one-to-one human-computer relationship.

interface—a piece of hardware, specifically designed to hook two other devices together. Usually some software is also required.

interpreter—a piece of system software that executes a program written in a high-level language directly. While useful for interactive computing, this system is too slow for most serious programming. Contrast with compiler.

interrupt—a signal that tells the CPU that a task must be done immediately. The registers are pushed to the stack, and a routine for the interrupt is branched to. When finished, the registers are popped from the stack and the main program continues.

I/O—acronym for input/output. Refers to the transfer of data.

I/O module—the printed circuit board that is the termination for field wiring of I/O devices.

I/O rack—a chassis which contains I/O modules.

I/O scan—the time required for the programmable controller processor to monitor all inputs and control all outputs. The I/O scan repeats continuously.

isolated I/O module—a module which has each input or output electrically isolated from every other input or output on that module. That is to say, each input or output has a separate return wire.

iteration—one pass through a given set of instructions.

J

jack—a socket, usually mounted on a device, which will receive a plug (generally mounted on a wire).

Josephson effect—the tunneling of electron pairs through a thin insulating barrier between two superconducting materials.

K

K—abbreviation for kilo. In computer terms 1024, in loose terms 1000.

Karnaugh map—a truth table that shows a geometrical pattern of functional relationships for gating configurations; with this map, essential gating requirements can be recognized in their simplest form.

L

ladder diagrams—an industry standard for representing control logic relay systems.

language—a set of symbols and rules for representing and communicating information (data) among people, or between people and machines.

large scale integration—LSI. Any integrated circuit which has more than 100 equivalent gates manufactured simultaneously on a single slice of semiconductor material.

latching relay—a relay with 2 separate coils, one of which must be energized to change the state of the relay; it will remain in either state without power.

leakage current—in general, the undesirable flow of current through or over the surface of an insulating material or insulator; the alternating current that passes through a rectifier without being rectified.

leakage flux—magnetic lines of force that go beyond their intended space.

least significant bit—the rightmost bit in a binary value, representing 2^0.

least significant byte—refers to the lowest position digit of a number. The rightmost byte of a number or character string.

LIFO—acronym for Last In First Out. Most CPUs maintain a "stack" of memory. The last data pushed onto the stack is the first popped out.

light emitting diode—LED. A semiconductor diode that converts electric energy efficiently into spontaneous and noncoherent electromagnetic radiation at visible and near infrared wavelengths of electroluminescence at a forward biased pn junction.

light pen—a device that senses light, interfaced to the computer for the purpose of drawing on the CRT screen.

line—in communications, describes cables, telephone lines, etc., over which data is transmitted to and received from the terminal.

line driver—an integrated circuit specifically designed to transmit digital information over long lines—that is, extended distances.

line printer—a high-speed printing device that prints an entire line at one time.

linear circuit—a network in which the parameters of resistance, inductance, and capacitance are constant with respect to current or voltage, and in which the voltage or current of sources is independent of or directly proportional to the outputs.

linearity—the relationship that exists between two quantities when a change in one of them produces a direct proportional change in the other.

location—a storage position in memory.

logic—a means of solving complex problems through the repeated use of simple functions which define basic concepts. Three basic logic functions are AND, OR, and NOT.

logic diagram—a drawing which represents the logic functions AND, OR, NOT, etc.

logic level—the voltage magnitude associated with signal pulses representing ones and zeroes (1s and 0s) in binary computation.

logical shift—a type of shift in which an operand is shifted right or left, with a zero filling the vacated bit position.

loop—a set of instructions that executes itself continuously. If the programmer has the presence of mind to provide for a test, the loop is discontinued when the test is met, otherwise it goes on until the machine is shut down.

loop counter—one way to test a loop. The counter is incremented at each pass through the loop. When it reaches a certain value, the loop is terminated.

low—a logic signal voltage. The computer senses this as a binary 0.

lsb—see least significant bit.

LSI—acronym for Large Scale Integration. An integrated circuit with a large number of circuits such as a CPU. See chip.

M

machine code—refers to programming instructions that are stored in binary and can be executed directly by the CPU without any compilation, interpretation, or assembly.

machine language—the primary instructions that were designed into the CPU by the manufacturer. These instructions move data between memory and registers, perform simple adding in registers, and allow branching based on values in registers.

macro—a routine that can be separately programmed, given a name, and executed from another program. The macro can perform functions on variables in the program that called it without disturbing anything else and then return control to the calling program.

magnetoresistor—magnetic field controlled variable resistor.

magnitude—the absolute value, independent of direction.

mainframe—refers to the CPU of a computer. This term is usually confined to larger computers.

mantissa—the fractional portion of a floating-point number.

matrix—a two-dimensional array of circuit elements, such as wires, diodes, etc., which can transform a digital code from one type to another.

memory—the hardware that stores data for use by the CPU. Each piece of data (bit) is represented by some type of electrical charge. Memory can be anything from tiny magnetic doughnuts to bubbles in a fluid. Most microcomputers have chips that contain many microscopic capacitors, each capable of storing a tiny electrical charge.

memory module—a processor module consisting of memory storage and capable of storing a finite number of words (e.g., 4096 words in a 4K memory module). Storage capacity is usually rounded off and abbreviated with K representing each 1024 words.

metal oxide semiconductor—MOS. A metal insulator semiconductor structure in which the insulating layer is an oxide of the substrate material; for a silicon substrate the insulator is silicon oxide.

micro electronics—refers to circuits built from miniaturized components and includes integrated circuits.

microprocessor—an electronic computer processor section implemented in relatively few IC chips (typically LSI) which contain arithmetic, logic, register, control, and memory functions.

microsecond—μs. One millionth of a second: $1 \times 10-^{6}$ or 0.000001 second.

millisecond—μs. One thousandth of a second: $10-^{3}$ or 0.001 second.

minuend—the number from which the subtrahend is subtracted.

mixed number—a number consisting of an integer and fraction as, for example, 4.35 or (binary) 1010.1011.

mnemonic—a short, alphanumeric abbreviation used to represent a machine-language code. An assembler will take a program written in these mnemonics and convert it to machine code.

modem—MOdulator/DEModulator. An I/O device that allows communication over telephone lines.

module—an interchangeable plug-in item containing electronic components which may be combined with other interchangeable items to form a complete unit.

monitor—1) a CRT 2) a short program that displays the contents of registers and memory locations and allows them to be changed. Monitors can also allow another program to execute one instruction at a time, saving programs and disassembling them.

MOS—see metal oxide semiconductor.

MOSFET—metal oxide semiconductor field effect transistor.

most significant bit—the leftmost bit in a binary value, representing the highest-order power of two. In two's complement notation, this bit is the sign bit.

most significant byte—the highest-order byte. In the multiple-precision number A13EF122H, A1H is the most significant byte.

msb—see most significant byte.

multiple-precision numbers—multiple-byte numbers that allow extended precision.

multiplexing—a method allowing several sets of data to be sent at different times over the same communication lines, yet all of the data can be used simultaneously after the final set is received. For example, several LED displays, each requiring four data lines, can all be written to with only one group of four data lines. The same concept is used with communication lines.

multiplicand—the number to be multiplied by the multiplier.

multiplicand register—the register used to hold the multiplicand in a machine-language multiply.

multiplier—the number that is multiplied against the multiplicand. The number "on the bottom."

N

NAND—an acronym for NOT AND. A Boolean logic expression. AND is performed, then NOT is performed to the result.

n-channel—a conduction channel formed by electrons in an n-type semiconductor, as in an n-type field-effect transistor.

negation—changing a negative value to a positive value, or vice versa. Taking the two's complement by changing all ones to zeros, all zeros to ones, and adding one.

nesting—putting one loop inside another. Some computers limit the number of loops that can be nested.

network—a collection of electric elements, such as resistors, coils, capacitors, and sources of energy, connected together to form several interrelated circuits. A collection of computer terminals interconnected to a host CPU.

noise—extraneous signals; any disturbance which causes interference with the desired signal or operation.

non-volatile memory—a memory that does not lose its information while its power supply is turned off.

normalization—converting data to a standard format for processing. In floating-point format, converting a number so that a significant bit (or hex digit) is the first bit (or four bits) of the fraction.

NOT—a Boolean operator that reverses outputs (1 becomes 0, 0 becomes 1). This is the one's complement.

NPN **transistor**—a junction transistor having a p-type base between an n-type emitter and an n-type collector; the emitter should then be negative with respect to the base and the collector should be positive with respect to the base.

n-type semiconductor—an extrinsic semiconductor in which the conduction electron density exceeds the hole density.

O

object code—all of the machine code that is generated by a compiler or assembler. Once object code is loaded into memory it is called machine code.

octal—refers to the base 8 number system, using digits 0–7.

octal-dabble—conversion of an octal number to decimal by multiplying by eight and adding the next octal digit, continuing until the last (rightmost) digit has been added.

OEM—Original Equipment Manufacturer.

off-line—describes equipment or devices which are not connected to the communications line.

offset value—a value that can be added to an address. Most addressing modes allow an offset value.

off-the-shelf—a term referring to software. A generalized program that can be used by a greater number of computer owners, so that it can be mass produced and bought off-the-shelf.

Ohm—the unit of resistance of a conductor such that a constant current of one ampere in it produces a voltage of one volt between its ends.

Ohm's law—a fundamental rule of electricity; states that the current in an electric circuit is inversely proportional to the resistance of the circuit and is directly proportional to the electromotive force in the circuit. In its strictest sense, Ohm's law applies only to linear constant-current circuits.

on-line—a term describing a situation where one computer is connected to another, with full handshake, over a modem line.

on-line operation—operations where the programmable controller is directly controlling the machine or process.

operands—the numeric values used in the add, subtract, or other operation.

OR—a Boolean logic function. If at least one of the lines tested is high (binary 1), the answer is high.

oscillation—any effect that varies periodically back and forth between two values, as in the amplitude of an alternating current.

output—the current, voltage, power, driving force, or information which a circuit or a device delivers. The terminals or other places where a circuit or device can deliver energy.

output devices—devices such as solenoids, motor starters, etc., that receive data from the programmable controller.

overflow—a condition that exists when the result of an add, subtract, or other arithmetic operation is too large to be held in the number of bits allotted.

overflow flag—a bit in the microprocessor used to record an overflow condition for machine-language operation.

overlay—a method of decreasing the amount of memory a program uses by allowing sections that are not in use simultaneously to load into the same area of memory. The new routine destroys the first routine, but it can always be loaded again if needed. Usually used in system programs.

oxide—an iron compound coating on tapes and disks that allows them to be magnetized so that they can be read by electrical devices and the information converted back to machine code.

P

padding—filling bit positions to the left with zeros to make a total of eight or sixteen bits.

page—refers to a 256 (2 to the 8th power) word block of memory. How large a word depends on the computer. Most micros are eight-bit word machines. Many chips do special indexed and offset addressing on the page where the program counter is pointing and/or on the first page of memory.

parallel—describes a method of data transfer where each bit of a word has its own data line, and all are transferred simultaneously.

parallel circuit—an electric circuit in which the elements, branches (having elements in series), or components are connected between two points, with one of the two ends of each component connected to each point.

parallel operation—type of information transfer whereby all digits of a word are handled simultaneously.

parallel output—simultaneous availability of two or more bits, channels, or digits.

parameter—a variable or constant that can be defined by the user and usually has a default value.

parity—a method of checking accuracy. The parity is found by adding all the bits of a word together. If the answer is even, the parity is 0 or even. If odd, the parity is 1 or odd. The bit sometimes replaces the most significant bit and usually sets a flag.

parity bit—an additional bit added to a memory word to make the sum of the number of 1s in a word always even or odd as required.

parity check—a check that tests whether the number of 1s in an array of binary digits is odd or even.

partial product—the intermediate results of a multiply. At the end, the partial product becomes the whole product.

partial product register—the register used to hold the partial results of a machine-language multiply.

passivation—growth of an oxide layer on the surface of a semiconductor to provide electrical stability by isolating the transistor surface from electrical and chemical conditions in the environment; this reduces reverse-current leakage, increases breakdown voltage, and raises power dissipation rating.

passive element—an element of an electric circuit that is not the source of energy, such as a resistor, inductor, or capacitor.

PC—see programmable controller.

PC board—see printed circuit board.

p-channel—a conduction channel formed by holes in a p-type semiconductor, as in a p-type field effect transistor.

peripheral devices—a generic term for equipment attached to a computer, such as keyboards, disk drives, cassette tapes, printers, plotters, speech synthesizers.

permeability—a factor, characteristic of a material, that is proportional to the magnetic induction produced in a material divided by the magnetic field strength given by the equation:

$$m = \frac{\text{magnetic induction (gauss)}}{\text{magnetizing field (oersteds)}}$$

permutation—arrangements of things in definite order. Two binary digits have four permutations: 00, 01, 10, and 11.

PILOT—a simple language for handling English sentences and strings of alphanumeric characters. Generally used for CAI.

PL/1—an acronym for programming language 1. A programming language used by very large computers. It incorporates most of the better features from other programming languages. Its power comes from the fact that bits can be manipulated from the high-level language.

plotter—a device that can draw graphs and curves and is controlled by the computer through an interface.

port—a single addressable channel used for communications.

P-N junction—a region of transition between p-type emitter and n-type semiconducting regions in a semiconductor device.

PNP transistor—a junction type transistor having an n-type base between a p-type emitter and a p-type collector.

positional notation—representation of a number where each digit position represents an increasingly higher power of the base.

precision—the number of significant digits that a variable or number format may contain.

print buffer—a portion of memory dedicated to holding the string of characters to be printed.

printed circuit board—a piece of plastic board with lines of a conductive material deposited on it to connect the components. The lines act like wires. These can be manufactured quickly and are easy to assemble the components on.

processor—a unit in the programmable controller which scans all the inputs and outputs in a predetermined order. The processor monitors the status of the inputs and outputs in response to the user-programmed instructions in memory, and it energizes or de-energizes outputs as a result of the logical comparisons made through these instructions.

product—the result of a multiply.

program—a sequence of instructions to be executed by the processor to control a machine or process.

program panel—a device for inserting, monitoring, and editing a program in a programmable controller.

program scan—the time required for the programmable controller processor to execute all instructions in the program once. The program scan repeats continuously. The program monitors inputs and controls outputs through the input and output image tables.

programmable controller—PC. A solid state control system which has a user-programmable memory for storage of instructions to implement specific functions such as I/O control logic, timing, counting, arithmetic, and data manipulation. A PC consists of the central processor, input/output interface, memory, and programming device which typically uses relay-equivalent symbols. The PC is purposely designed as an industrial control system which can perform functions equivalent to a relay panel or a wired solid state logic control system.

PROM—Programmable Read Only Memory. A memory device that is written to once and from then on acts like a ROM.

protocol—a defined means of establishing criteria for receiving and transmitting data through communication channels.

pseudo code—a mnemonic used by assemblers that is not a command to the CPU, but a command to the assembler itself.

p-type semiconductor—an extrinsic semiconductor in which the hole density exceeds the conduction electron density.

punched-card equipment—peripheral devices that enable punching or reading paper punched cards that hold character or binary data.

Q

quotient—the result of a divide operation.

R

RAM—acronym for Random Access Memory. An addressable LSI device used to store information in microscopic flip-flops or capacitors. Each may be set to an ON or OFF state, representing logical 1 or 0. This type of memory is volatile, that is to say, memory is lost while power is off, unless battery backup is used.

read—to sense the presence of information in some type of storage, which includes RAM memory, magnetic tape, punched tape, etc.

real time clock—a clock in the sense that we normally think of one, interfaced to the computer.

record—a file is divided into records, each of which is organized in the same manner.

register—a fast-access memory location in the microprocessor. Used for holding intermediate results and for computation in machine language.

relative addressing—an address that is dependent upon where the program counter is presently pointing.

remainder—the amount of divident remaining after a divide has been completed.

residue—the amount of dividend remaining, part way through a divide.

resistor-transistor logic—RTL. One of the simplest logic circuits, having several resistors, a transistor, and a diode.

resolution—a measure of the smallest possible increment of change in the variable output of a device.

restoring divide—a divide in which the divisor is restored if the divide "does not go" for any iteration. A common microcomputer divide technique.

ROM—an acronym for Read Only Memory. Memory that is addressed by the bus, but can only be read from. If you tell the CPU to write to it, the machine will try, but the data is not remembered.

rotate—a type of shift in which data is recirculated right or left back into the operand from the opposite end.

rounding—the process of truncating bits to the right of a bit position and adding zero or one to the next higher bit position based on the value to the right. rounding the binary fraction 1011.1011 to two fractional bits, for example, results in 1011.11.

RPG—an acronym for Report Program Generator. A language for business that primarily reads data from cards and prints reports containing that data.

RS-232—an interface that converts parallel data to serial data for communications purposes. The output is universally standard.

rung—a grouping of PC instructions which controls one output. This is represented as one section of a logic ladder diagram.

S

scaled up—referring to a number which has been multiplied by a scale factor for processing.

scaling—multiplying a number by a fixed amount so that a fraction can be processed as an integer value.

scan time—the time necessary to completely execute the entire programmable controller program one time.

scientific notation—a standard form for representing any size number by a mantissa and power of ten.

self-diagnostic—the hardware and firmware within a controller which allows it to continuously monitor its own status and indicate any fault which might occur within.

semiconductor—a compound that can be made to vary its resistance to electricity by mixing it differently. Layers of this material can be used to make circuits that do the same things tubes do, but using much less electricity. Transistors and integrated circuits are made from semiconductive material and are called semiconductors.

semiconductor device—an electronic device in which the characteristic distinguishing electronic conduction takes place within a semiconductor.

sensor—a sensing element, a device which senses either the absolute value or the change in a physical quantity, and converts that change into a useful signal for an information-gathering system.

serial—a way of sending data, one bit at a time, between two devices. The bits are rejoined into bytes by the receiving device. contrast with parallel.

serial operation—type of information transfer within a programmable controller whereby the bits are handled sequentially rather than simultaneously, as they are in parallel operation. Serial operation is slower than parallel operation for equivalent clock rates. However, only one channel is required for serial operation.

series circuit—a circuit in which all parts are connected end to end to provide a single path for current.

shift and add—a multiply method in which the multiply is achieved by shifting of and addition of the multiplicand.

shift register—a program, entered by the user into the memory of a programmable controller, in which the information data (usually single bits) is shifted one or more positions on a continual basis. There are two types of shift registers: asynchronous and synchronous.

sign bit—sometimes the most significant bit is used to indicate the sign of the number it represents. 1 is negative (–) and 0 is positive (+).

sign extension—extending the sign bit of a two's complement number to the left by a duplication.

sign flag—a bit in the microprocessor used to record the sign of the result of a machine-language operation.

sign-magnitude—a nonstandard way of representing positive and negative numbers in microcomputers.

signed numbers—numbers that may be either positive or negative.

significant bits—the number of bits in a binary value after leading zeros have been removed.

significant digit—a digit that contributes to the precision of a number. The number of significant digits is counted beginning with the digit contributing the most value, called the most significant digit, and ending with one contributing the least value, called the least significant digit.

silicon controlled rectifier—SCR. A semiconductor rectifier that can be controlled; it is a pnpn four-layer semiconductor device that normally acts as an open circuit, but switches rapidly to a conducting state when an appropriate gate signal is applied to the gate terminal.

simulator—a computer that is programmed to mimic the action and functions of another piece of machinery, usually for training purposes. A computer is usually employed because it is cheaper to have the computer simulate these actions than to use the real thing. Airplane and power plant trainers are excellent examples.

sink—a device that drains energy off a system; a device that switches a load to an absorbing material, such as a ground.

software—refers to the programs that can be run on a computer.

solid state devices (semiconductors)—electronic components that control electron flow through solid materials such as crystals; e.g., transistors, diodes, integrated circuits.

SOS—silicon on sapphire. A semiconductor manufacturing technology in which metal oxide semiconductor devices are constructed in a thin single-crystal silicon film grown on an electrically insulating synthetic sapphire substrate.

source program—the program written in a language or mnemonics that is converted to machine code. The source program as well as the object code generated from it can be saved in mass storage devices.

special purpose logic—proprietary features of a programmable controller which allow it to perform logic not normally found in relay ladder logic.

SPOOL—acronym for Simultaneous Peripheral Output, On-Line. Used to overlap processing, typically, with printing.

stack—an area of memory used by the CPU and the programmer particularly for storage of register values during interrupt routines. See LIFO.

start-up—the time between equipment installation and the full operation of the system.

state—the logic 0 or 1 condition in programmable controller memory or at a circuit's input or output.

status register—the register that contains the status flags set and tested by the CPU operations.

stepper motor—a special motor in a disk drive that moves the read/write head a specific distance each time power is applied. That distance defines the tracks on a disk.

storage—see memory.

strip printer—a peripheral device used with a programmable controller to provide a hard copy of process numbers, status, and functions.

subroutine—a routine within a program that ends with an instruction to return program flow to where it was before the routine began. This routine is used many times from many different places in the program, and the subroutine allows you to write the code for that routine only once. Similar to a macro.

substrate—the physical material on which a microcircuit is fabricated; used primarily for mechanical support and insulating purposes; however, semiconductor and ferrite substrates may also provide useful electric functions.

subtract with carry—a machine-language instruction in which one operand is subtracted from another, along with a possible borrow from the next lower byte.

subtrahend—the number that is subtracted from the minuend.

successive addition—a multiplication method in which the multiplicand is added a number of times equal to the multiplier to find the product.

surge—a transient variation in the current and/or potential at a point in the circuit.

synchronous shift register—shift register which uses a clock for timing of a system operation and where only one state change per clock pulse occurs.

syntax—the term is used exactly as it is used in English composition. Every language has its own syntax.

system—a collection of units combined to work as a larger integrated unit having the capabilities of all the separate units.

system software—software that the computer must have loaded and running to work properly.

T

table—an ordered collection of variables and/or values, indexed in such a way that finding a particular one can be done quickly.

tape reader—a unit which is capable of sensing data from punched tape.

Teletype^{TM}—a peripheral electromechanical device for entering or outputting a program or data in either a punched paper tape or printed format.

termination—1) the load connected to the output end of a transmission line 2) the provisions for ending a transmission line and connecting to a bus bar or other terminating device.

text editor—see word processor.

thumbwheel switch—a rotating numeric switch used to input numeric information to a controller.

timer—in relay-panel hardware, an electromechanical device which can be wired and preset to control the operating interval of other devices. In the programmable controller a timer is internal to the processor, which is to say it is controlled by a user-programmed instruction. A timer instruction has greater capability than any hardware timer. Therefore, programmable controller applications do not require hardware timers.

time sharing—refers to systems which allow several people to use the computer at the same time.

track—a concentric area on a disk where data is stored in microscopic magnetized areas.

transducer—a device used to convert physical parameters, such as temperature, pressure, and weight into electrical signals.

translator package—a computer program which allows a user program (in binary) to be converted into a usable form for computer manipulation.

transistor—an active component of an electronic circuit consisting of a small block of semiconducting material to which at least three electrical contacts are made, usually two closely spaced rectifying contacts and one ohmic (non-rectifying) contact; it may be used as an amplifier, detector, or switch.

transistor-transistor logic—TTL. A logic circuit containing two transistors, for driving large output capacitances at high speed. A family of integrated circuit logic. (Usually 5 volts is high or 1 and 0 volts is low or 0; 5V = 1, 0V = 0).

Triac^{TM}—a General Electric trademark for a gate controlled semiconductor switch designed for alternating current power control; with phase control of the gate signal, load current can be varied over a range from 5 percent to 95 percent of full power.

truncation—the process of dropping bits to the right of a bit position. Truncating the binary fraction 1011.1011 to a number with fraction of two bits, for example, results in 1011.10.

truth table—a table defining the results for several different variables and containing all possible states of the variables.

TTL—see transistor-transistor logic.

TTY—an abbreviation for Teletype.

two's complement—a standard way of representing positive and negative numbers in microcomputers.

U

unsigned numbers—numbers that may be only positive; absolute numbers.

utility—a program designed to aid the programmer in developing other software.

UV erasable PROM—an ultraviolet erasable PROM is a programmable read-only memory which can be cleared (set to 0) by exposure to intense ultraviolet light. After being cleared, it may be reprogrammed.

V

variable—a labeled entity that can take on any value.

volatile memory—a memory that loses its information if the power is removed from it.

volt—the unit of potential difference or electromotive force in the meter-kilogram-second system, equal to the potential difference between two points for which 1 coulomb of electricity will do 1 joule of work in going from one point to the other.

voltage—potential difference or electromotive force capable of producing a current; measured in volts.

voltage drop—the voltage developed across a component or a conductor by the flow of current through the resistance or impedance of the component or conductor.

von Neumann, John (1903–1957)—Mathemetician. He put the concept of games, winning strategy, and different types of games into mathematical formulae. He also advanced the concept of storing the program in memory as opposed to having it on tape.

W

weighted value—the numerical value assigned to any single bit as a function of its position in the code word.

word—a grouping or a number of bits in a sequence that is treated as a unit and is stored in one memory location. If the CPU works with 8 bits, then the word length is 8 bits. Common word sizes are 4, 8, 12, 16, and 32. Some are as large as 128 bits.

word processor—a computer system dedicated to editing text and printing it in various controllable formats. See editor.

write—to store in memory or on a mass storage device.

X

XOR—a Boolean function. Acronym for eXclusive OR. Similar to OR but answer is high (1) if and only if one line is high.

Z

zero flag—a bit in the microprocessor used to record the zero/non-zero status of the result of a machine-language instruction.

zero page—refers to the first page of memory.

INDEX

INDEX

INDEX COMPILED BY NAN MCCARTHY

The real value of your computer lies in your ability to use it. The capabilities of the TRS-80* are incredible if you have the information which will help you get the most from it. Little of this information is available in your instruction books.

The *Encyclopedia for the TRS-80* will teach you how to get the most from your computer. In addition to a wealth of programs which are ready for you to use, reviews of accessories and commercially available programs, you will also learn how to write your own programs or even modify commercial programs for your own specific use.

The *Encyclopedia* is packed with practical information, written and edited for the average TRS-80 owner, not the computer scientist. You will find it interesting and valuable.

Wayne Green
Publisher

ISBN 0-88006-034-4